COMPUTER DATA BASE

ASSESSMENT OF MASONRY BRIDGES

by

LAWRENCE SIHWA

A thesis submitted in fulfilment of the requirements

for the Degree of Doctor of Philosophy in

Electrical Engineering

University of Edinburgh

1987

DECLARATION

I hereby declare that this thesis was composed by myself, and the original works and results reported were obtained solely by myself, unless otherwise stated.

Edinburgh, June, 1987

L. Sihwa

DEDICATION

TO MY MOTHER

and

MY LATE FATHER

CONTENTS

DECLARATION i

DEDICATION ii

CONTENTS iii

ACKNOWLEDGEMENT ix

ABSTRACT x

CHAPTER 1

INTRODUCTION

1.0 BACKGROUND 1

1.1 MAINTENANCE CONSIDERATIONS 2

1.1.1 Safety Considerations 2

1.1.2 Economic Considerations 2

1.1.3 Other Considerations 3

1.2 BRIDGE MANAGEMENT SYSTEMS 3

1.2.1 Decision Support Systems 4

1.3 PROJECT WORK AND SUMMARY OF THESIS 5

CHAPTER 2

STONE MASONRY STRUCTURES

2.0 HISTORICAL BACKGROUND 8

2.1 BENDING AND COMPRESSION STRUCTURES

..... 11

2.2 COMPONENT PARTS OF MASONRY ARCH

AND THEIR FUNCTIONS 14

2.2.1 Arch Ring 14

2.2.2 Spandrel Walls and Fill Material..... 16

2.2.3 The Arch Barrel 16

2.2.4 Abutments, Piers and Foundations..... 16

2.3 DEFECTS AND DETERIORATION

OF MASONRY STRUCTURES 17

2.3.1 Deterioration of Spandrel Walls 17

2.3.2 Change in Shape of Arch 18

2.3.3 Cracks in Barrel 18

2.3.4 Downward Displacement of Individual Stones 21

2.3.5 Leaching 21

2.3.6 Spalling 22

2.3.7 Defects in Fill Material 22

2.3.8 Defects in Foundations 22

2.4 MECHANISMS OF COLLAPSE OF AN ARCH 27

2.5 INTERACTION OF COMPONENT PARTS OF MASONRY ARCH .....

REFERENCES AND BIBLIOGRAPHY 29

CHAPTER 3

CURRENT METHODS OF ASSESSMENT AND MAINTENANCE

OF STONE MASONRY STRUCTURES

3.0 BACKGROUND 30

3.1 ASSESSMENT TECHNIQUES 31

3.2 PRESENT ASSESSMENT METHODS 32

3.2.1 Visual Inspection and Coring 32

3.2.2 Modified MEXE Method 33

3.2.3 Static Loading 36

3.2.4 Dynamic Signature 38

3.2.5 Sonic Assessment 40

3.2.6 Tale Tales 44

3.3 TECHNIQUES FOR REPAIR AND MAINTENANCE
OF MASONRY BRIDGES 45
3.3.1 Filling of Cavities by Injection 45
3.3.2 Improvements of Drainage and Water Evacuation Systems 47
3.3.3 Waterproofing of the Extrados 41
3.3.4 Repointing and Restoration of Masonry 48
3.3.5 Installation of Steel Cross Ties 48
3.3.6 Installation of Steel Ties 53
3.3.7 Repairs to Abutments and Piers 53
3.3.8 Repairs to Arch Barrels 54
3.3.9 Repair of Spandrell Walls 56
3.3.10 Repair to Road Surfacing 57
3.4 OVERVIEW 57
REFERENCES AND BIBLIOGRAPHY 57
CHAPTER 4
DATA BASES AND DATA BASE MANAGEMENT SYSTEMS
4.0 BACKGROUND 59
4.1 DATA PROCESSING 60
4.1.1 The Traditional Approach 60
4.1.2 The Data Base Approach 62
4.2 DATA BASE MANAGEMENT ARCHITECTURE 64
4.3 DATA BASE MODELS 66
4.3.1 Hierarchical Model 67
4.3.2 Network Model 68
4.4 DATA BASE NAVIGATION IN FORMATTED SYSTEMS 60
4.5 EVALUATION OF FORMATTED SYSTEMS 70
REFERENCES AND BIBLIOGRAPHY 70

CHAPTER 5

THE RELATIONAL MODEL

5.0 INTRODUCTION 72

5.1 BASIC CONCEPTS OF A RELATIONAL DATA BASE 73

5.2 PROPERTIES OF THE RELATIONAL MODEL 73

5.3 DATA MANIPULATION LANGUAGES
FOR THE RELATIONAL MODEL 74

5.3.1 Relational Calculus 75

5.4 RELATIONAL DATA BASE DESIGN 76

5.4.1 Normalisation 78

5.4.2 Third Normal Form 81

5.4.3 Boyce/Codd Normal Form (BCNF) 81

5.5 EVALUATION OF THE MODELS 83

5.6 EXAMPLES ON MASONRY BRIDGES 84

REFERENCES AND BIBLIOGRAPHY 85

CHAPTER 6

THE INGRES RELATIONAL DATA BASE MANAGEMENT SYSTEM

6.0 BACKGROUND 87

6.1 USING THE INGRES RELATIONAL DATA
BASE MANAGEMENT SYSTEM 88

6.2 QUEL AND INGRES UTILITY COMMANDS 89

6.3 EQUEL 91

6.4 THE INGRES PROCESS STRUCTURE 91

6.4.1 UNIX 91

6.4.2 Invoking INGRES from UNIX 92

6.4.3 Indirect Interaction with INGRES 93

6.5 INGRES STORAGE STRUCTURES 94

6.5.1 Heap 94

6.5.2 Hash 95

6.5.3 Isam 95

6.5.4 Secondary Indices 95

6.5.5 Compression 96

REFERENCES AND BIBLIOGRAPHY 96

CHAPTER 7

DEVELOPMENT OF THE INTERFACE

7.0 BACKGROUND 99

7.1 BASIC STRUCTURE OF USER INTERFACE 100

7.2 ROLE OF THE THREE PROCESSES 102

7.2.1 Parent Process 102

7.2.2 INGRES (Child 1) and Child 2 102

7.3 PROCESS CONTROL 102

7.4 INTERPROCESS COMMUNICATION 104

7.5 OPERATION OF INTERFACE 105

7.5.1 Communication Between Parent and INGRES 105

7.5.2 Communication Between INGRES and Child 2 107

REFERENCES AND BIBLIOGRAPHY 108

CHAPTER 8

APPLICATION OF THE SYSTEM

8.0 BACKGROUND 109

8.1 STRUCTURE OF SYSTEM 110

8.1.1 Information Retrieval 110

8.1.2 Associative Process 112

8.2 SYSTEM CHARACTERISTICS 113

8.3 SYSTEM PERFORMANCE 113

8.4 SYSTEM MODIFICATION 115

8.5 RUNNING THE SYSTEM 117

8.6 DATA BASE DESIGN FOR THE SYSTEM 117

CHAPTER 9

DEMONSTRATION RUN

9.1 SHOW RELATIONS IN A DATA BASE 119

9.2 LIST DOMAIN NAMES AND FORMATS 120

9.3 PRINT OUT CONTENTS OF A RELATION 121

9.4 VIEW CONTENTS OF QUERY BUFFER 122

9.5 INFORMATION RETRIEVAL 123

9.6 DECISION SUPPORT SOFTWARE 126

9.7 CREATING AND MAINTAINING A

A DATA BASE USING INGRES.....129

9.7.1 Create an Empty Relation 130

9.7.2 Forming a Relation From Existing Relations 131

9.7.3 Destroy a Relation 131

9.7.4 Erase Contents of a Relation 132

9.7.5 How to Copy Whole Relations to INGRES 132

9.7.6 Modify System Relations 133

9.7.7 To Destroy a data base 134

9.7.8 Storage Structures in INGRES 134

9.7.9 Quit Sub Menu to Main Menu 135

CHAPTER 10

DISCUSSION

10.0 CONCLUDING REMARKS 136

10.1 IMPLICATIONS TO CIVIL ENGINEERS 138

10.3 DISCUSSION 139

APPENDIX 3..... 141

MEXE METHOD

APPENDIX 6A..... 149

QUEL COMMANDS

APPENDIX 6B..... 152

INGRES UTILITY COMMANDS

APPENDIX 8..... 157

LISTING OF CURRENT RELATIONS

APPENDIX 10..... 162

LISTING OF THE INTERFACE BETWEEN THE USER

AND THE INGRES RELATIONAL DATA BASE SYSTEM

ACKOWLEDGEMENTS

The author wishes to thank Professor J. Mavor, head of the Department of Electrical Engineering for placing the facilities of this Department at his disposal at all times. The author is grateful to Professor A. W. Henry, head of Department of Civil Engineering for his help in the project work.

The author is particularly grateful to Dr. H.W. Whittington and Dr. G.C. Coghill for their supervision. Their advise was invaluable in carrying out the project work as well as in the preparation of this thesis.

The author is indebted to the staff at the Highway Department, the British Rail Civil Engineering Department, and the Scottish Development Department, all in Edinburgh, for allowing me to look into their records and for the time they spent discussing the project with me.

The author also wishes to thank his mother, brother and sisters for their moral and financial support at all times.

The financial support of the Science and Engineering Research Council is gratefully acknowledged.

ABSTRACT

This thesis is concerned about the development of computer data base management system for the assessment of masonry bridges.

The various techniques of assessment and remedial measures of masonry bridges are outlined, their shortcomings described. A justification for an alternative method of assessment is given.

A review of computer data base management systems is carried out. The reasons for adopting data base management systems is given as well as the reasons for choosing a particular type of data base management system.

The common faults associated with masonry structures are described and the problems of identifying these faults are described. The part played by the individual components of a masonry arch bridge is given and the significance of faults on the individual components of the structure is described.

A detailed description of the type, in general of the data base system chosen is given followed by a detailed description of a special case of the type chosen, which is the system that was used for the project.

A description of how the system was developed is given followed by the way the system operates.

A detailed description of how the system can be used is then put forward and the problems associated with the development of the system are outlined.

Finally, a description of the implications of the system to the practising engineer is given.

CHAPTER 1

INTRODUCTION

1.0 BACKGROUND

Bridges form a key part of the highway, railway and aqueduct network because of their strategic location and of the undesirable consequences if they fail or if their capacity is impaired. Particular attention must therefore be given to the systematic assessment of bridges as an essential part of the surveillance and management of the transport system.

Like all engineering structures, bridges, for the most part, start to deteriorate from the moment they are built. Also, it is common experience that they are subjected to increased vehicular loading as they age often well above that for which they were conceived. In order to repair and maintain these structures, the engineer requires knowledge on their design details as well as their operational and maintenance histories. In the case of recently constructed structures, this information is normally available, but unfortunately, in the past, little or no thought was given to maintenance when designing structures and there was very little feedback from maintenance to design. Consequently, in assessing old structures for repair and maintenance, the engineer is faced with difficulties as a result of:

- non-availability of records for the structures;
- the structures are in themselves very old;
- non-uniform design and construction methods were used and
- some repair work has often been done but not documented.

1.1 MAINTENANCE CONSIDERATIONS

1.1.1 Safety Considerations

Failures of bridges in service do occur, but are fortunately rare, particularly those leading to personal injury. More common is accidental damage arising from impact by vehicles or vessels.

Linked with safety is the concept of serviceability. This entails that the bridge must to an acceptable degree of probability, perform its specified functions without undue expense in terms of both capital and maintenance costs. It should do so for the whole of its expected life without being non-operational (except for reasonable short maintenance overhauls) without detriment to its appearance and to public confidence in its safety.

1.1.2 Economic Considerations

It is a vital necessity to keep a close and systematic watch on the bridge structure and its constituent elements, in order to ensure that the necessary action is taken in time and most economically. In fact, if traffic has to be interrupted across a bridge as a result of the occurrence of serious damage before there has been time to plan repairs or reconstruction, the cost to the community can be of a very high order. For example, there have been incidences where the road user costs due to an unplanned and unexpected replacement of a bridge have been of the order of five times the cost of rebuilding the bridge, reference [3.4(1)].

In the United Kingdom the bridge stock is in excess of 120,000 representing a replacement cost of approximately £15,000 million. Several thousand of these bridges are stone masonry bridges, dating back, in most cases, for over two hundred years and presenting a major, and expensive, problem for their responsible Authorities. The

maintenance of stone masonry structures can be extremely expensive undertaking because of the following points:

- the replacement materials are expensive and often difficult to obtain;
- the maintenance personnel need to be highly skilled, hence wage costs are high;
- in many instances, when planned repairs are begun additional bridge faults become apparent when the structure is partly dismantled.

1.1.3 Other Considerations

In a significant number of cases, the bridges are listed structures with statutory obligations in terms of preservation, for example, for architectural interests. Hence, rather than consider replacement, the Engineer must maintain and repair. In addition, the legal liability with which authorities or even the individual bridge Engineer may be faced can be an important incentive for setting up or perfecting bridge maintenance schemes.

Even if there are no legal sanctions, the sight of neglected, deteriorating or damaged bridges may affect public confidence and subject the responsible bridge authority to severe criticism. Moreover, in an era that has witnessed on the one hand unprecedented increases in international transport, travel and tourism and on the other ever more complex and spectacular engineering structures, the publicity likely to be given to serious bridge accidents or failures may have an adverse influence on professional or national prestige.

1.2 BRIDGE MANAGEMENT SYSTEMS

Timely and economic planning and programming of remedial and preventative maintenance and repair work, or even bridge replacement, with the minimum effect upon traffic, are dependent upon systematic and detailed bridge inspection (see Chapter 3) and the expert assessment of data.

It is possible that with present assessment methods (see Chapter 3), some bridges may be selected for repair while more seriously damaged structures are missed. It follows that if maintenance scheduling could be optimised by making available to the Engineer responsible for their repair and maintenance, the collected expertise of practitioners over many years, there would be consequent attractive associated cost benefits. This would be both in terms of improved interpretation of the results of inspection and testing and of allowing the ranking of bridges in order of need for repair such that the most serious cases were dealt with first.

Hence, if software aids for repair and maintenance scheduling were made available to the engineer, they could provide the following advantages:

- more effective ranking;
- more effective repair strategy;
- reliable and safer transport routes.

1.2.1 Decision Support Systems

Effective access to expert knowledge is possible by consulting an expert, but even then it is often considered necessary to seek a second opinion. If the combined knowledge of many experts is available, the process becomes even more attractive, but, in practice this is very difficult to achieve or very expensive.

However, with appropriate software, computer Data Base Management can effectively

make it possible for an engineer to access the collected experience of many experts and of the archives of many bridge authorities.

To demonstrate the feasibility of the decision support software in maintenance and repair scheduling, the area chosen has been that of maintenance and repair of stone masonry bridges because of the following reasons:

- the structures are very old, in most cases over 200 years of age;
- no documentation exists for these structures;
- the structures are of non-standard design;
- assessment of the structures by quantitative means is not suitable because of the non-homogeneous nature of the materials of construction.
- archive data is available in different forms;
- much of the assessment is based on Engineer's experience and not on theory and
- testing is expensive and of limited value.

1.3 PROJECT WORK AND SUMMARY OF THESIS

The project work can be broken down into three major parts, viz:

(i) the generation of a data base of test inspection, repair and maintenance information relevant to stone masonry structures;

(ii) the development of a user friendly interface for retrieval of archived data from the data base;

(iii) the development of a basic decision support program to demonstrate the feasibility of the technique to the practising engineer and to allow future feedback from potential users;

Chapter 2 gives a historical background to the masonry arch, that is, how it came about and the reasons as to why there are several thousand masonry arch bridges in the United Kingdom. The various component parts of a masonry arch bridge and their respective functions are described in turn. This is followed by a description of the defects and types of deterioration that are common to masonry arch bridges.

Chapter 3 outlines the techniques currently employed for the assessment of stone masonry bridges as well as their shortcomings. The techniques described are: visual inspection and coring, modified Military Engineering Experimental Establishment (MEXE) method, static loading, dynamic signature and sonic assessment. The various methods for the repair and maintenance of stone masonry bridges are described.

Chapter 4 gives a review of data bases. The disadvantages associated with the traditional approach to data processing systems are outlined. The advantages of the data base approach to data processing over the traditional approach are outlined. Two data base models, the hierarchical model and the network model are described in turn and their drawbacks are outlined.

Chapter 5 describes the third data base model, the relational model. The advantages of this model over the other models described in Chapter 4 are outlined thereby providing a justification for using this type of model for the project work.

Chapter 6 looks in depth at the INGRES Relational Data Base Management System, that is, the relational data base model adopted for this project. This chapter describes the essential features of INGRES that were used in the project work.

Chapter 7 describes how an interface between INGRES and the user was developed as well as how it operates. The three major processes that make up the system are described, that is, how they were developed and how they communicate with each other.

Chapter 8 describes how the user interface described in Chapter 7 was put into use. The two major components of the system, that is, information retrieval and associative processes are described in turn. The major problems that were encountered in the development of the system are outlined with the ways in which they were overcome. This chapter also gives a detailed description of how the system could be modified to suit future user needs. This is followed by a description of how the system could be initially run.

Chapter 9 gives examples of typical investigations through the data base using the various facilities which have been developed, with explanations which would enable the user to use the data base unaided, with reference only to this thesis. Examples of each of the facilities developed are given in chapter 9.

Chapter 10 is a discussion of the project work. The implications to Civil Engineers of the project work are outlined. This chapter also outlines the advantages that the system would offer to the user over a conventional data base system.

CHAPTER 2

STONE MASONRY STRUCTURES

2.0 HISTORICAL BACKGROUND

Our distant forebears in many parts of the globe, struggling to overcome the transportation problems of their primitive trail-worlds, invented not only the simple beam bridge but also the far more sophisticated, even ultramodern forms of suspension and cantilever.

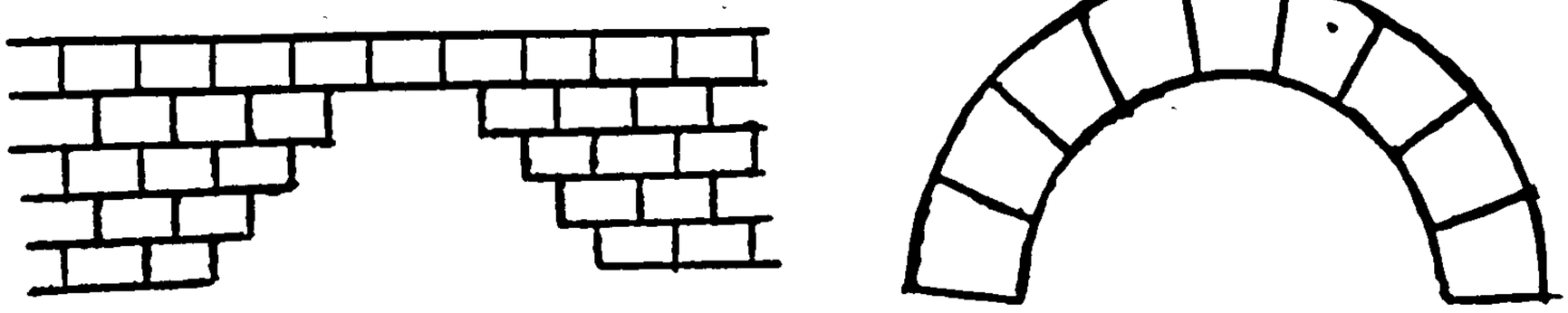
In the interior of South America men first learned to swing across a chasm on a vine, like monkeys. In China cantilevers were built by extending heavy timbers outward from a solid store abutment. For early man, with his limited access to materials and his limited means of refining them, these bridge forms, had only very restricted value. They served for narrow crossings under favourable circumstances.

Civilisation demanded something better. Above all, the invention of the wheel, with its dramatic train of carts, wagons, roads, highways, merchants, wealth, towns, and cities, brought the problem of river crossing to the fore.

The invention that solved the bridge problem of ancient civilisation ranks second only to the wheel itself. It is the arch. How this marvel came into being is as deep a mystery as the origin of the wheel. Engineers discount the older guess that man built arches in imitation of nature, for the natural arch formed by erosion is structurally quite different from the stone arch. Archaeologists have found that the arch appeared in tombs and underground temples long before it was used as a bridge. Recent excavations have disclosed underground vaults going back to fourth millennium B.C. at Ur and elsewhere in ancient Sumer, the earliest Tigris-Euphrates civilisation. Egyptians, too knew vaulting by the year 3000 B.C.

The Greeks had used beam and column construction for many centuries. However, the only materials abundantly available were stone, brick or combinations of the two. Arches were built of these materials which are suitable for structures in compression and could resist buckling.

The Sumerians and Babylonians apparently had the false arch (Figure 2.1 (a)) at a very early date, and perhaps derived the true arch (Figure 2.1 (b)) from it. The false arch, built of overlapping bricks laid horizontally and held together by mortar, will stand, but will not carry a load. A true arch, on the other hand, will sustain an enormous weight even without any mortar.



(a) (b)

Figure 2.1 False and True Arch

Restricted to what amounted to a decorative role in tombs, temples, and palaces, the arch existed for at least two thousand years before it was ever used as a bridge. For its serious application, the stone arch, like many other Greek, Persian and Egyptian inventions, awaited the coming of the gifted Romans.

To appreciate the Roman achievements one should first take some note of the basic problems involved in bridging a wide, deep river. To bridge a river with five stone arches required four piers in the stream, two of them near the middle. Since the bottom of a river is mud, the problem was how to build a pier on a mud river bottom. Supporting a heavy stone arch demanded a pier of considerable dimensions. The longer the arch span, the thicker the pier. Roman arch bridges, almost without excep-

tion, had semicircular profiles, with the width of each intermediate pier 1/4 to 1/3 of the clear opening (span). The Romans usually made their arches from 15 to 27 meters in span, and their piers from 5 to 11 meters thick. All the arches of a bridge were not necessarily the same length, more frequently the centre span or two centre spans were longer than the outside spans.

The Romans arrived at this semicircular profile by trial and error, and by the same process discovered that piers of the above proportion proved most satisfactory. Also, because they relied on empirical proportioning they were ignorant of the structural principles governing the stability of arches.

The old Roman semicircular stone arch persisted for century after century and was only superseded by the flat, elliptical arch on the eve of the metal bridge. Such an arch has a profoundly different structural effect from that of the semicircular arch. Where the semicircular arch presses down in a wholly vertical direction, the segmental arch introduced the element of horizontal thrust. In many locations the semicircular arch demanded steeply inclined roadways, while a flatter arch could keep the roadway level enough for easy wagon passage. Also, the segmental arch required fewer piers in the stream than the semicircular arch thereby offering less obstruction to navigation and at the same time freer passage to floodwaters.

The stone used in the construction of masonry arches was mostly ashlar; that is, rectangular blocks properly tooled on the external face only. In some cases the arches were built of local shistos stone, commonly called "whin", which quarries in thin flat pieces long enough to form the thickness of an arch but with very irregular edges and faces. The most regular of these stones were chosen for the facing of the arch on the elevations and irregular pieces were used to make up the rest of the arch. Since carriage was usually at least as much if not more than the price of the stone itself, in the majority of cases, the stone used was from local quarries. Alternatively, in some cases,

it was imported usually by water since this was often easier and cheaper than by land.

In general, masonry was laid in lime mortar, but sometimes, especially when exposed to continual wet conditions, it was laid in wax and pitch or in resin applied hot.

The eighteenth century was a century of revolutions. One of these revolutions was the industrial revolution. As commerce grew, and manufacturing in turn grew to feed commerce, and basic industries, especially coal mining, grew to feed manufacturing, the rapidly expanding economic machine demanded new, quicker, better, transport. Hence there are several thousand masonry bridges in the United Kingdom. These structures are several hundred years old, while the design life of many structures today is approximately 75 - 100 years, which is an indictment on our current technology.

2.1 BENDING AND COMPRESSION STRUCTURES

One of the most important structural forms is the beam. This type of structural form transmits forces by bending. However, force transmission by bending is not efficient in comparison to axial force transmission structures such as trusses.

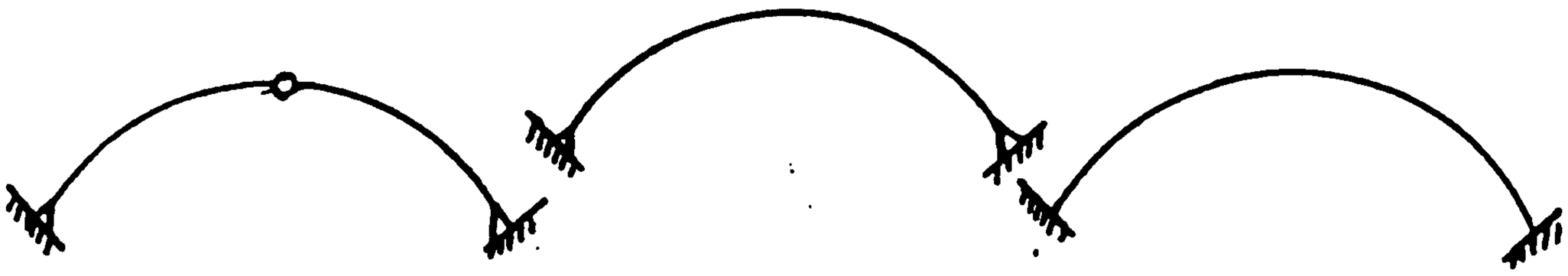
The two simple structural forms suited for carrying forces by compression alone are the column and the arch. A column is a straight member loaded along its centroidal axis with a compressive load. Except when it is extremely short, the column is less efficient than a tensile member because it has the tendency to buckle when compressed.

A third form is the dome which can be created by revolving a parabolic or circular arch about a vertical axis through its highest point.

There are three types of arches depending on the support conditions: three-hinged, two-hinged, and hingeless (fixed) arches, Figure 2.2.

The 3-hinged arch is the only arch type where movement of the support will not pro-

duce stresses in the arch. If a fixed arch is used, the foundations must be very rigid because slight translation or rotation of the ends will produce substantial stresses in the arch. Two hinged arches are not affected by foundation rotation but are sensitive to



Three-hinged Two-hinged Hingeless

Figure 2.2 Arch Types

2.1.1 Comparison of Beam and Arch

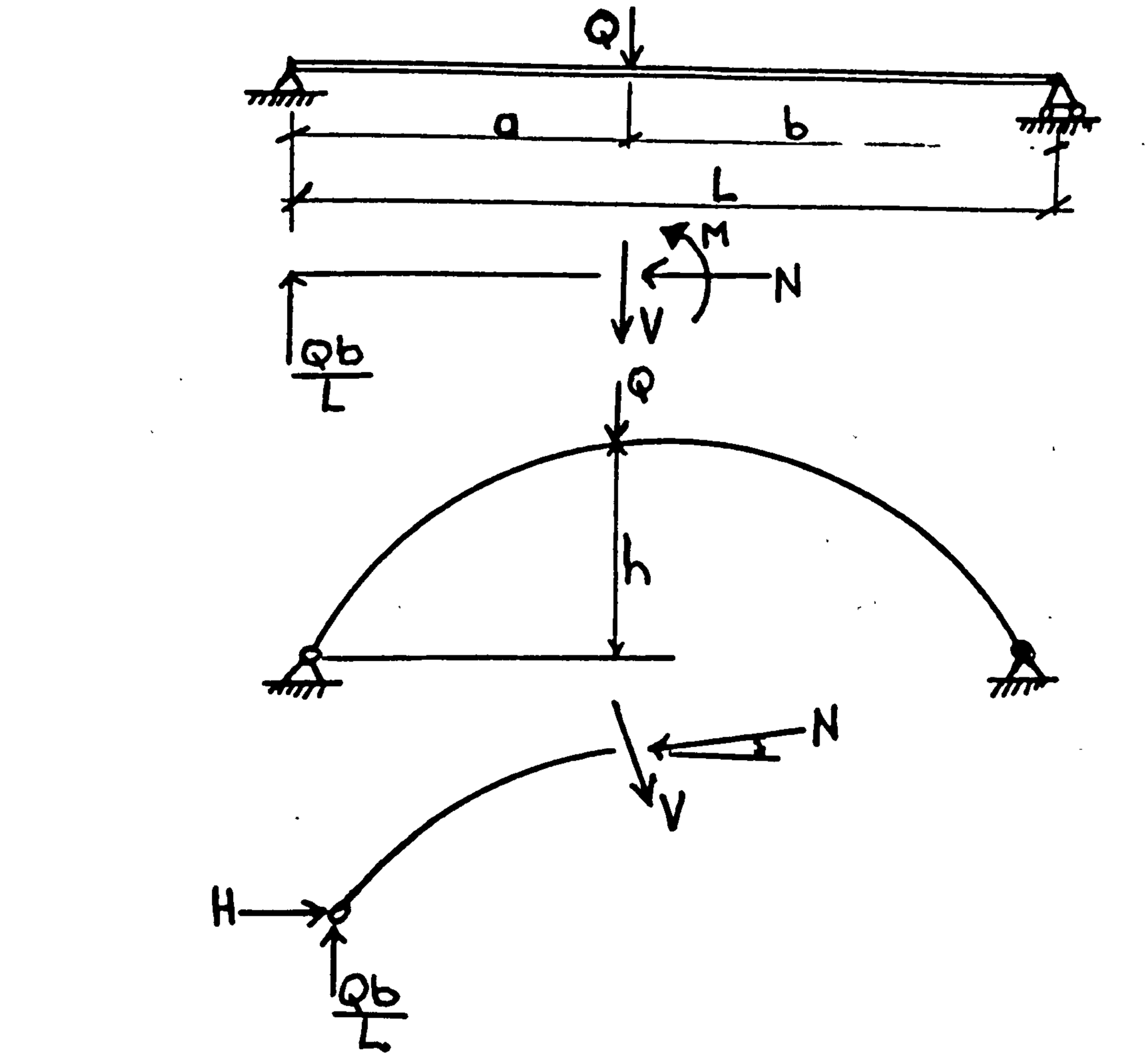


Figure 2.3 Comparison of Beam and Arch

Force transmission by bending is not efficient in comparison to axial force transmission. In general, arches provide a marked reduction in bending moment but at the expense of both large horizontal forces at the supports. All arches develop horizontal reactions that produce moments opposite to those developed by the vertical reactions and loads. This behaviour can be best illustrated by considering the state of internal forces in a beam and an arch over the same span length L (Figure 2.3). At a point just to the left of the load Q the bending moment M, shear force V and axial force N in the beam are as follows:

$$M = \frac{Q.a.b}{L}$$

$$V = \frac{Q.b}{L}$$

$$N = 0$$

At the same point in the arch, the corresponding values are:

$$M = \frac{Q.a.b}{L} - H.h$$

$$V = \frac{Q.b.\cos\theta}{L} - H.\sin\theta$$

$$H = \frac{Q.b}{L}.\sin\theta + H.\cos\theta.$$

It is thus evident that both M and V are lower in the arch than in the beam.

Bending members place special requirements on materials since the material must be capable of carrying both tensile and compressive stresses of about the same magnitude. This creates no problem with most metals, except for the possibility of buckling tendencies on the compression side of the beam, but it rules out the use of a material weak in tension, such as masonry.

2.2 COMPONENT PARTS OF MASONRY ARCH AND THEIR FUNCTIONS

Figure 2.4 shows the details of a masonry arch bridge. The following sections describe the various components of the bridge and their respective functions. Photographs I and II show single and double arch bridges respectively.

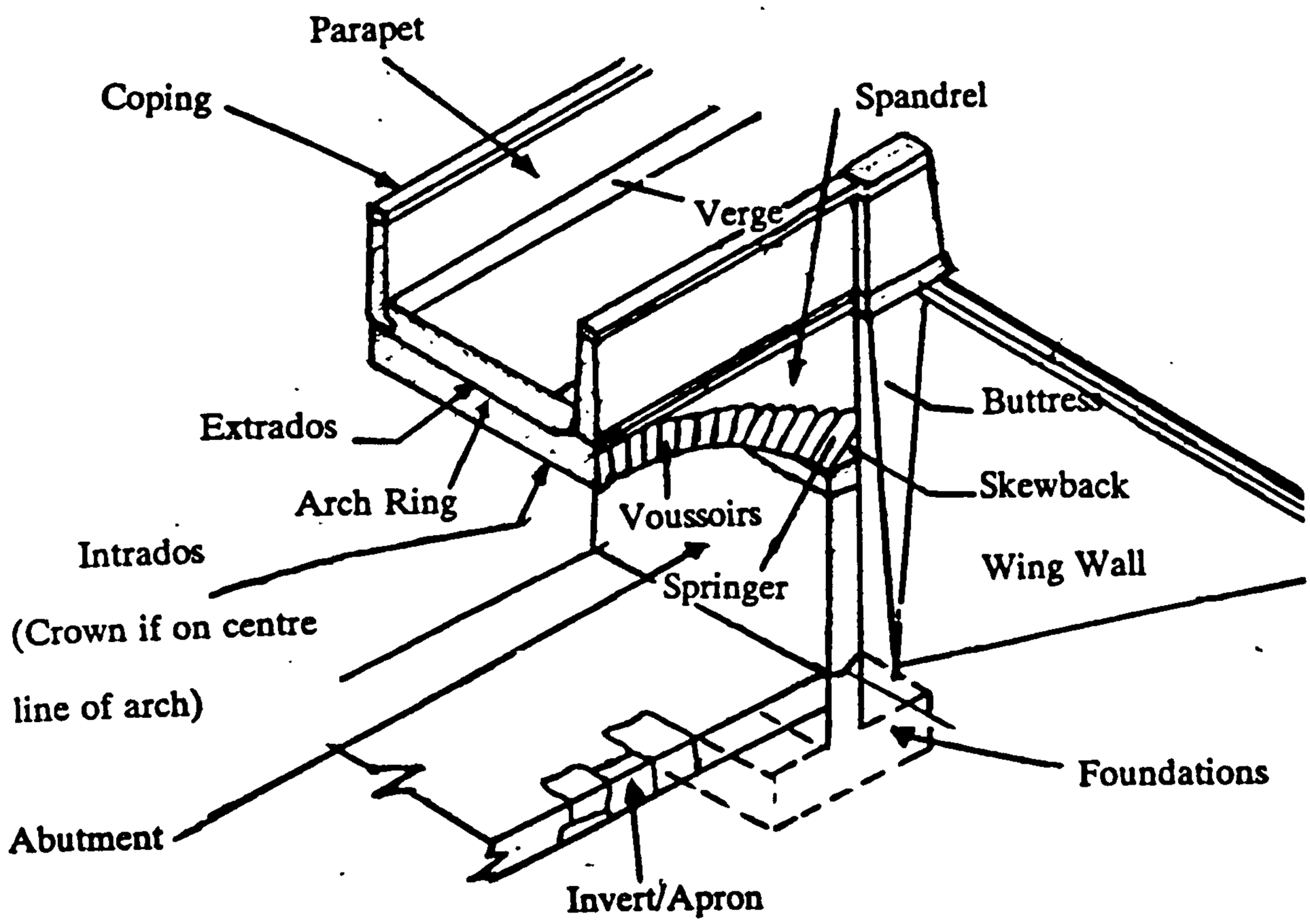


Figure 2.4 Details of a Masonry Arch Bridge

2.2.1 Arch Ring

The arch ring forms the basic structural component of the bridge and is made up of voussoirs (Figure 2.4). For a large span arch the voussoirs would normally be carefully cut and a minimum of mortar used for their assembly while roughly cut stones, with thicker mortar would be used for smaller arches.

Photograph I General View of Single Span Masonry Bridge

Photograph II General View of Double Span Masonry Bridge

2.2.2 Spandrel Walls and Fill Material

Spandrel walls are built on the arch rings on the two faces of the bridge to restrain the infilling material. It is because of this function that spandrel walls are a contributing factor to the carrying capacity of the bridge. The fill for small bridges is usually a composition of rubble of earth or gravel or hoggin built up to the desired height to carry the road surface. The fill material contributes the larger proportion of the dead weight of the bridge. Although a non structural component of the bridge, the infill provides a medium for the distribution of the applied loading to the extrados of the arch.

In the case of larger bridges, a series of walls is constructed over the haunches and these walls have similar functions as those described above for the infill.

2.2.3 The Arch Barrel

This is the continuous prismatic structure that is formed by neighbouring arch rings. As already mentioned in Section 2.0, the masonry is not well cut behind voussoirs, although this is seldom the case for larger spans. Hence, in most cases, the voussoirs have varying axial lengths implying that the neighbouring parallel arch rings interlock and are not necessarily independent. The varied axial lengths also imply that the thickness of the visible arch ring on the external face of a bridge is not necessarily the thickness of the barrel under the roadway. This should be noted in determining the thickness of the arch barrel (see Chapter 3, Section 3.2.2).

2.2.4 Abutments, Piers and Foundations

Abutments and piers provide the resistance to the horizontal thrust of the arch; and hence the end supports of the arch and at the same time distribute the load from the arch to the foundations.

2.3 DEFECTS AND DETERIORATION OF MASONRY STRUCTURES

Considering that masonry arches are at present carrying live load much in excess of that for which they were designed, these bridges have given excellent services in terms of strength and durability. Nonetheless, many defects and types of deterioration do occur due to degradation of bridge with time or to the accidental occurrence such as impact, flooding or excessive loading. Also, recent increases in traffic loads and densities has led to defects and deterioration of these bridges.

The following sections outline some of the commonly found defects, deformations and types of deterioration found in masonry bridges and their respective causes.

Photographs III - VI show some of the defects and deformations found in masonry bridges.

2.3.1 Deterioration of Spandrel Walls

Apart from deterioration due to such factors as weather, loss of pointing etc., spandrel walls deteriorate because of lateral pressures developed in the infill by dead and live loading as a result of the following factors:

- lateral spread of the infilling as it is subjected to traffic loading. This is more pronounced in bridges where traffic approaches the edges of the bridge, which is often the case in bridges without footways;
- frost heave which occurs when the water trapped in the fill material freezes thereby expanding and exerting very high pressure on the walls;
- direct impact on walls by traffic which is more likely in bridges without footway to prevent traffic approaching the edges of the bridge.

The overall effect of the above mentioned lateral forces is the outward movement of

the walls which may be in the form of:

- bulging of wall;
- rotation of wall from arch barrel;
- sliding of wall from arch barrel;
- displacement of wall together with the arch ring.

Besides the outward movement of the walls due to lateral pressures, cracking at the haunches may take place as an indication of some form of flexibility of the arch ring over the centre half of the span.

2.3.2 Change in Shape of Arch

This is often a result of a partial failure of the arch which may be due to impact from traffic or from movements at the abutments.

2.3.3 Cracks in Barrel

In the following section, settlement is the vertical movement of the ground due to compressibility of soil while subsidence is the movement of the earth due to collapse of soil containing cavities, usually due to mineral mining.

The cracks in the barrel are mainly distinguishable by their orientation and their location The following cracks may be observed on the barrel:

- longitudinal cracks close to the edge of the arch, caused by the factors already mentioned above (Section 2.3.1) as shown in Figure 2.5 (a) and Photograph III.
- differential settlement of the abutments leads to longitudinal cracking of the barrel, Figure 2.5 (b). There is cause for alarm if these cracks are large (>3mm), as this indicates that the arch has subdivided into narrow

segments which are less efficient of transferring the load than the unsegmented arch.

- lateral cracks - these may result from the reasons already discussed above (Section 2.3.1).
- subsidence at the sides of the abutments causes diagonal cracking of the barrel, Figure 2.5 (c). Such cracks normally originate at the vicinity of the sides of the arch at its springing (Figure 2.1) spreading upwards towards the crown of the bridge. When pronounced (>3mm) reference [3.4], such cracks signal a dangerous state of affairs in the overall condition of the bridge.

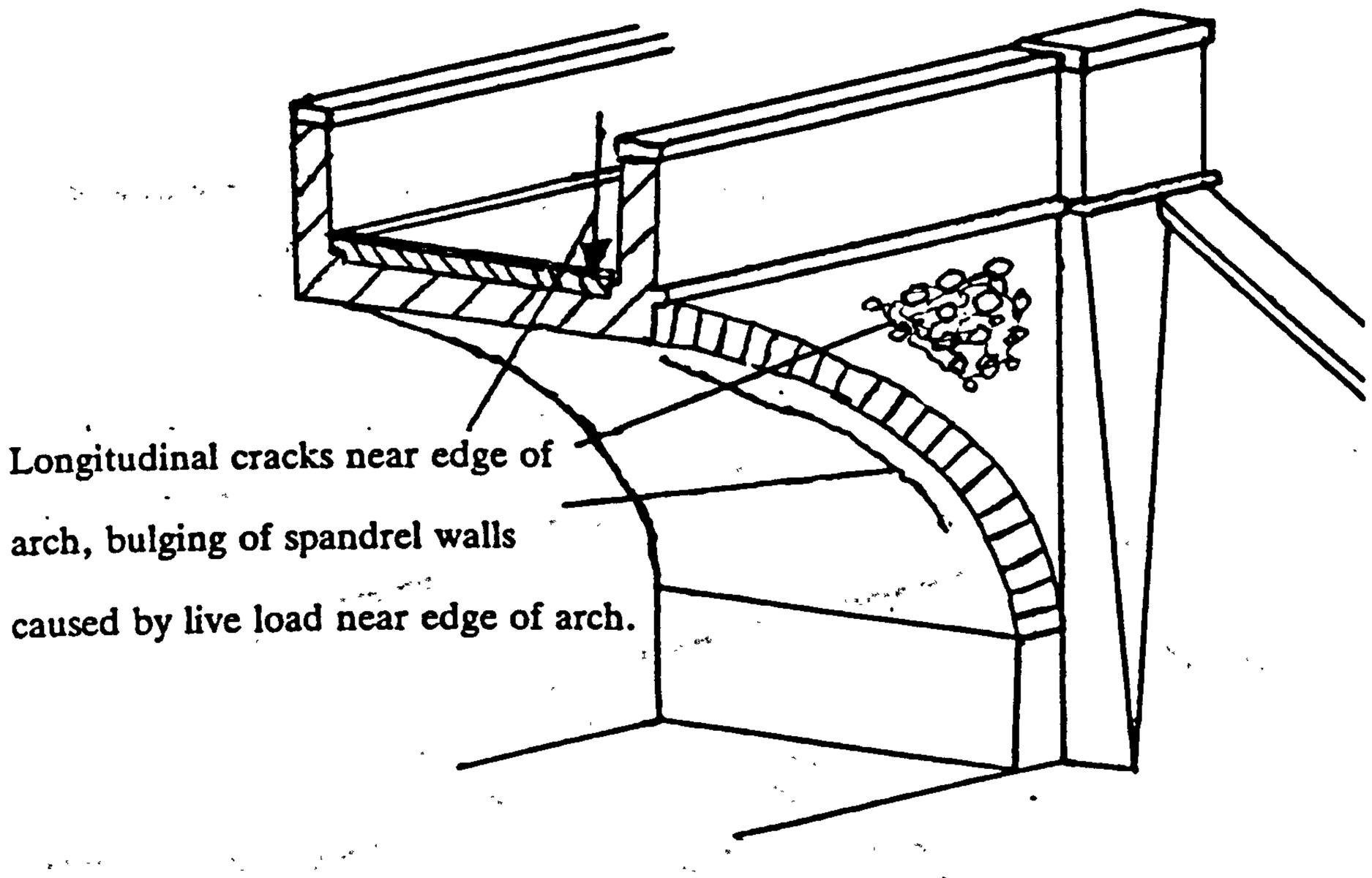


Figure 2.5 (a) Defects in Masonry Arches

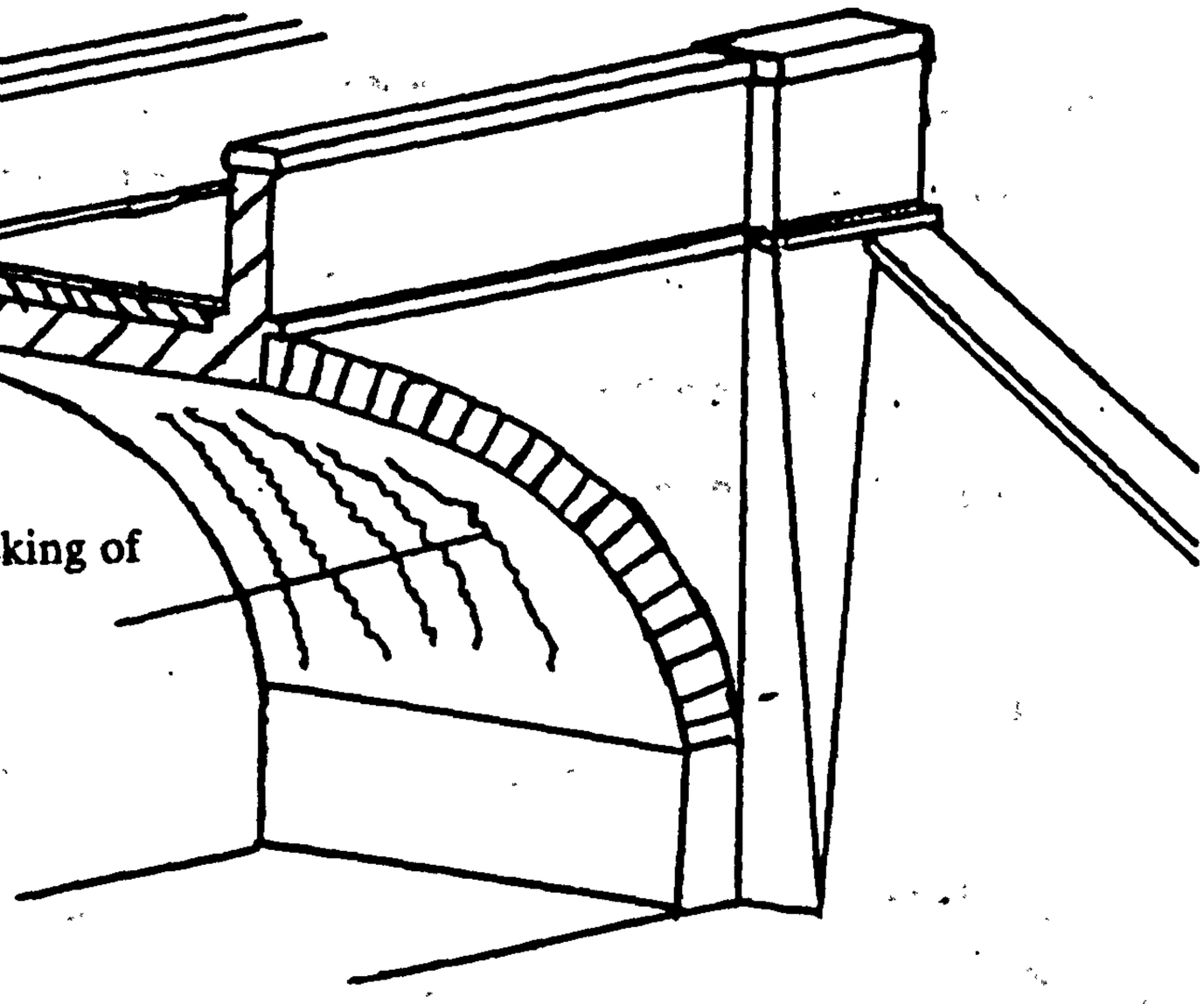


Figure 2.5 (b) Defects in Masonry Arches

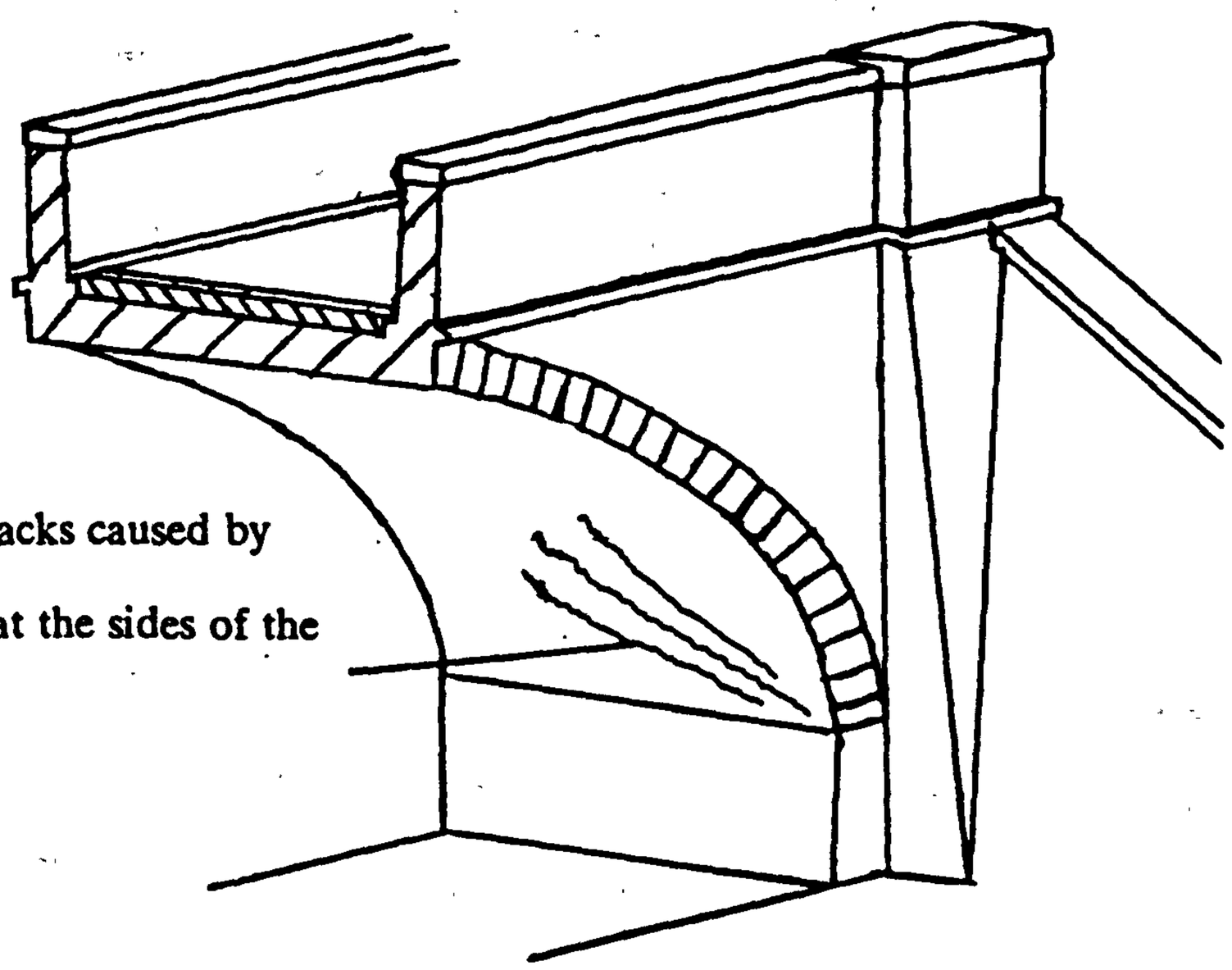


Figure 2.5 (c) Defects in Masonry Arches

2.3.4 Downward Displacement of Individual Stones

The uneven cut of masonry as already mentioned in Section 2.0, implies that some masonry projects above the barrel and is therefore subjected to concentrated loads such as from services. This is more likely to occur at the crown where the infill is minimal and may therefore be inadequate. This results in the downward displacement of individual stones as shown in Figure 2.5 (d).

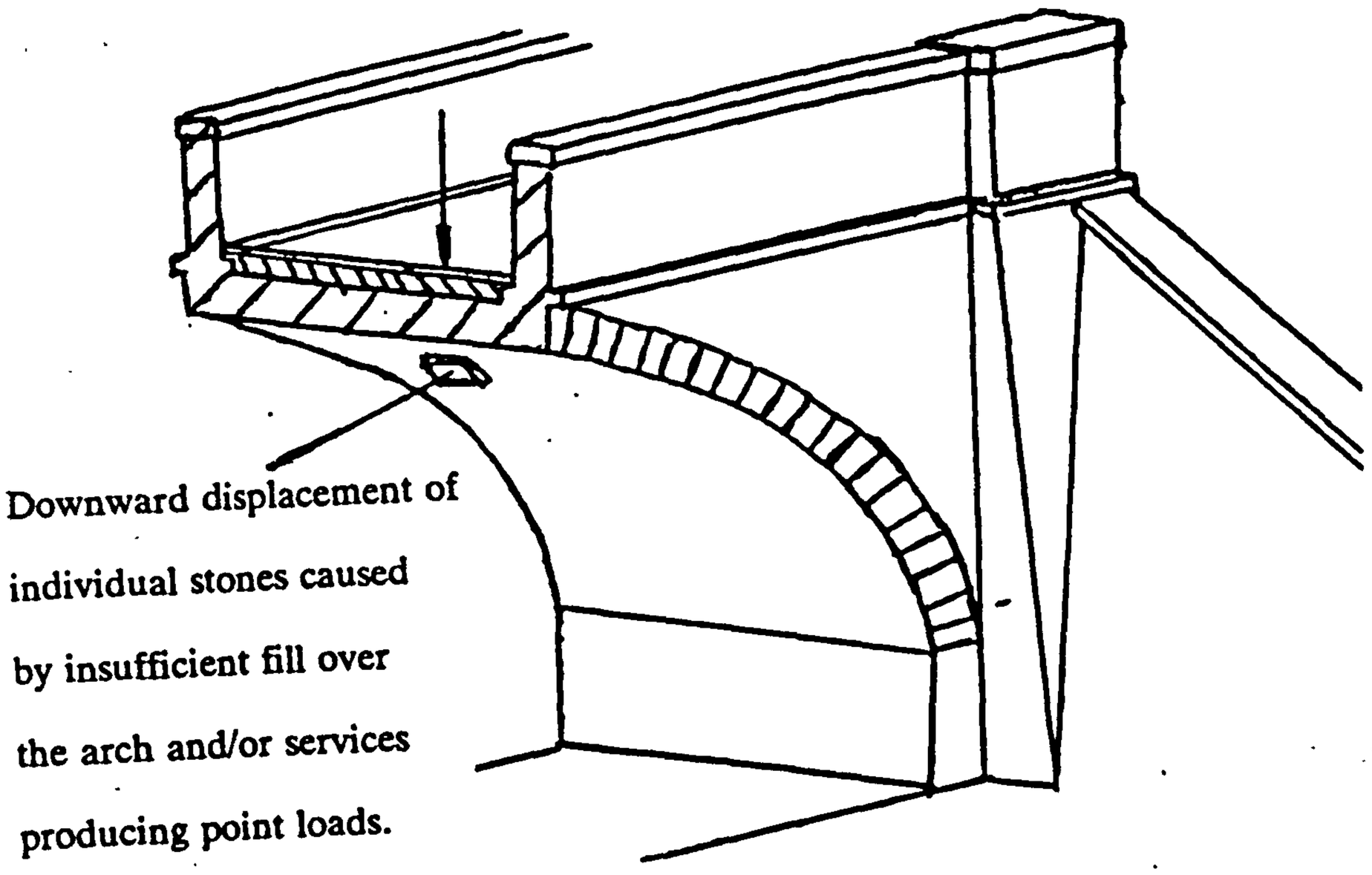


Figure 2.5 (d) Defects in Masonry Arches

2.3.5 Leaching and Spalling

The leakage or seepage of water through joints may dissolve out lime at the joints or the fill material. This phenomenon known as leaching is the removal of material usually lime, from concrete or masonry by percolation of water.

This form of deterioration is more likely to occur in arches which are continuously wet or are indicative of frequent penetration of dampness.

Spalling is the detachment of fragments, usually flaky, from a larger mass which may result from: a blow, or the action of weather (onion weathering), or internal pressure due to expansion in icy weather of water trapped in cracks of masonry, or impact from traffic.

2.3.6 Defects in Fill Material

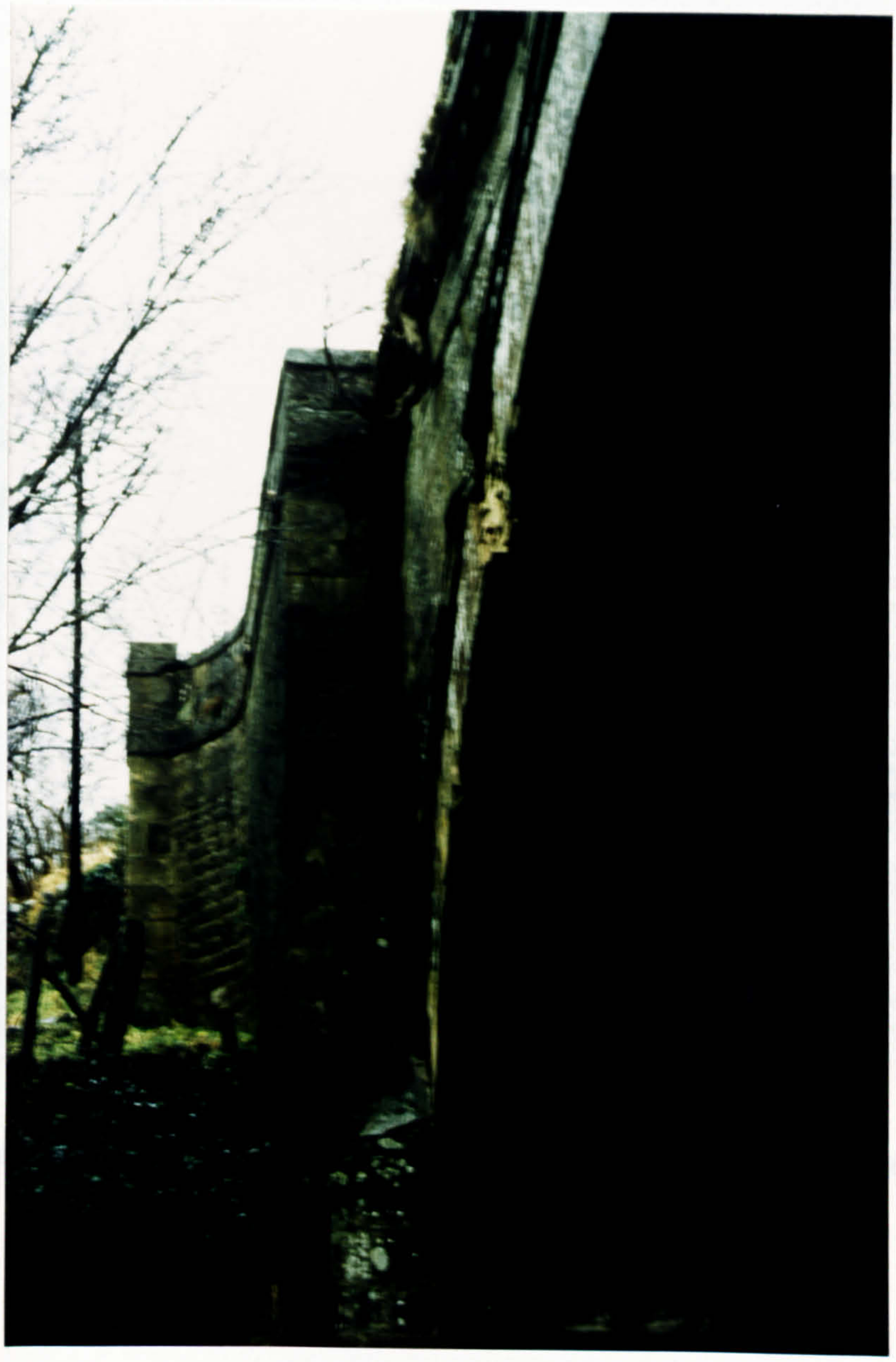
Defects and deterioration may take place in the fill itself, on account of poor quality material, lack of compaction, or loss of fill (Section 2.3.5). These defects may sometimes be detected by "tracking" becoming visible in the road deck surfacing. In some of the old bridges, strengthening has been achieved by the removal of the infill material and replacing it with concrete. Saturation of the infill material due to poor drainage leads to a reduction in the shear strength of the infill which may result in the overall deterioration of the bridge.

2.3.7 Defects in Foundations

Foundations of masonry arch bridges do tend to be highly susceptible to deterioration. This is because, during the times they were constructed, it was a much greater comparative task than would be the case in today's standards to excavate to a suitable stratum or drive piles to bear the foundations. As a result, less consideration was given to scour and settlement than would be the case in today's standards.

Photograph III Longitudinal Cracking Close to Edge of Arch.

Photograph IV Separation of Arch Ring From Spandrel Walls.



Photograph V Spalling of Voussoirs

Photograph VI Gouging of Stones Caused by Passing River Traffic

2.4 MECHANISMS OF COLLAPSE OF AN ARCH

In Figure 2.6 (b) an idealised semi-circular arch is supposed to be acted upon by its own weight and by a single point load. As this point load is imagined to increase slowly in magnitude, the self weight of the arch will have less effect on the shape of the funicular polygon; in the limit, the thrust line will consist of the two straight lines shown. For the particular dimensions sketched in Figure 2.6 (b) it is evident that a sufficiently large point load cannot be supported by two straight thrust lines lying wholly within the masonry as the point load increases in magnitude, a stage will be reached when the arch collapses by the mechanism of four hinges sketched in Figure 2.6 (c).

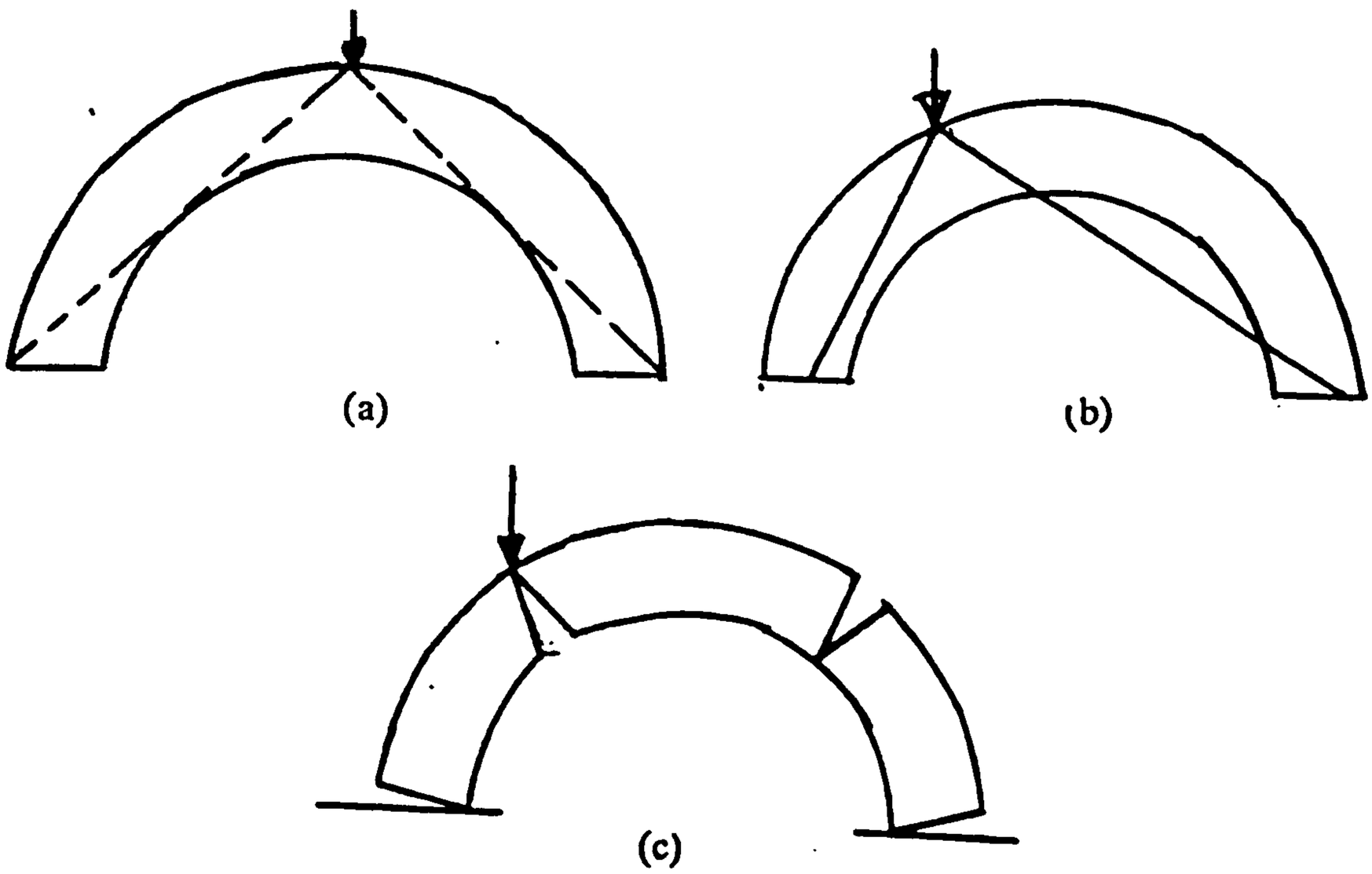


Figure 2.6 Mechanisms of Collapse of an Arch.

The idealised arch has been drawn with very particular proportions; in Figure 2.6 (a) it will be seen that straight thrust lines can be drawn within the masonry to support a point load placed at the crown. This implies that the arch can carry a point load of any

intensity at the crown (provided the strength of the material against crushing is not exceeded). The four-bar chain, Figure 2.6 (c) is the basic mechanism of collapse of an arch.

A full account on the mechanisms of collapse of arches can be found in Heyman, reference [2.2]

2.5 INTERACTION OF COMPONENT PARTS OF MASONRY ARCH

As already discussed above, masonry bridges comprise various components which interact with each other and their environment to varying degrees. The overall performance of the bridge depends on the condition of the individual elements as well as their overall interaction. To ensure the satisfactory performance of the bridge, it is essential that the bridge be monitored frequently to detect any defects and deterioration that might have occurred. Chapter 3 describes the techniques currently used for the assessment of stone masonry bridges as well as typical remedial measures that could be used on masonry bridges.

REFERENCES AND BIBLIOGRAPHY

[2.1] Gies J., "Bridges and Men", Cassell and Company Ltd., London, 1964, pp 2 - 33.

[2.2] Heyman J., "The Masonry Arch", Ellis Horwood Series in Engineering Science,1979, pp 50 - 62.

[2.3] Hopkins H.J., "A Span Of Bridges", Newton Abbot, David and Charles, 1970, pp 71 - 93.

[2.4] Ruddock T., "Arch Bridges and Their Builders", 1735-1835, Cambridge University Press, pp 112 - 140.

[2.5] White R.N., Gergely P.,and Sexsmith R.G., "Structural Engineering, Introduction to Design Concepts and Analysis", John Wiley and Sons, Inc., 2nd ed., 1972, pp 172 - 180.

CHAPTER 3

CURRENT METHODS OF ASSESSMENT AND MAINTENANCE OF STONE MASONRY BRIDGES

3.0 BACKGROUND

Checking the adequacy of an existing structure sometimes becomes necessary as a result of:

- defects in design and construction;
- deterioration with time or in service;
- accidental damage or collapse;
- for purchase, insurance, or legal purposes;
- change of use;
- future safety or serviceability.

The overall purpose of bridge assessment is to ensure that the bridge performs its specified functions without:

- running into undue maintenance costs;
- totally or partially disrupting the services;
- destroying public confidence in its safety.

The assessment process requires a knowledge of the integrity of the structure. To obtain this knowledge entails inspection, analysis and calculation followed by interpretation and Engineering judgement. Because of the complexity and heterogeneous

nature of bridge construction, calculation and analysis are normally applied to the arch barrel. Assessment of the spandrel walls, substructures, foundations and wing walls is normally by qualitative judgements of information obtained from inspection.

3.1 ASSESSMENT TECHNIQUES

For all bridges, attempts must be made to assess all relevant features of the construction. Several complementary test methods may have to be used, but even then the final decision will still rest with the professional Engineer responsible for the bridge.

Considerations of safety and serviceability of a structure and its components are closely related to assessment techniques. These techniques cover a wide spectrum ranging from assessment by the naked eye up to complex electronic-based techniques.

The testing of masonry structures has traditionally been limited to the direct determination of the mechanical properties of samples, taken as representatives of the structure under consideration. Recent developments in non-destructive test techniques such as non-destructive sonic testing (Whittington, 1984) which although limited in its value on its own, provides additional information to traditional methods and have the added advantages that they can be applied in-situ to full scale structures.

Partially destructive or destructive tests are also employed but only to determine the compliance of physical, mechanical, chemical ot other properties with the requirements of standards or specifications as well as instruments used for research purposes. Their significance lies on providing information which assists in the development of other new techniques or the improvement of existing ones. Because of their nature, non-destructive assessment techniques are mainly used in bridge inspection, since here in situ testing is essential, in many cases.

3.2 PRESENT ASSESSMENT METHODS

None of the techniques described below should be considered as the best one but a combination of the different methods will usually give acceptable conclusions.

3.2.1 Visual Inspection and Coring

This is the most commonly used method of assessment of stone masonry structures. Visual inspection requires considerable skill and background knowledge and is an essential feature of any assessment programme.

A bridge inspector with experience may accomplish a satisfactory visual assessment using virtually, no tools or equipment except for his naked eye. Because of limitations on what the naked eye can in most cases achieve, standard inspection tools such as pocket knifes, magnifying glass, cameras etc., are used to aid the naked eye.

In carrying out a visual inspection, the engineer should take consideration of not only the condition of the individual components of the bridge but also of the structure as an entity and of specific features which are indicative of structural deterioration.

The main points to be observed to ascertain the general state of the bridge by visual inspection are:

- deterioration and crumbling of exposed surfaces due to weathering leading in more severe cases, to spalling and splitting of stones;
- opening of joints and movements of supports as such movements will cause the loss of bedding mortar between components of arch and, in severe cases, to displacement of stone blocks;
- drainage of infill materials between spandrel walls. High porosity materials have the potential of storing a substantial volume of water, increasing undesirably the load on the arch and also accelerating the deterioration of

materials, particularly under frost action;

- accumulation of debris and vegetation. Because of the likelihood of the accumulation of soil in various parts of the structures, a good deal of vegetation can be supported on the bridge. The root system can however, cause damage;
- overall alignment and geometry - undesirable changes of shape can sometimes be detected visually before the structure is unserviceable.

Apart from visual inspection, the most common present assessment technique is coring the bridge. This involves the removal of sections of the bridge for inspection and if necessary, mechanical testing for the strength of the material and engineering properties. Such testing is generally limited to a few points on the bridge and, although it provides information, this information is limited to the location cored.

3.2.2 Modified MEXE Method

The method is based on Pippard's pre-war papers reported in Civil engineer in war [3.5]. Pippard confined his analysis to that for a single point load at mid-span. Pippard was aware that in theory an arch rib is weakest under the action of a point load at quarter points rather than at the crown. However, he argued reasonably for the use of the result for the central load on the grounds of the distribution of the load from the road surface through the fill to the arch proper. If a conventional 90 wedge angle is taken for the dispersion of the load, then the effective width of the arch when the load acts at the crown is 2h (Figure 8.1). A greater width of arch will be available to carry the point load at quarter span, since the load will be dispersed through a greater thickness of fill.

It should be noted, however, that this method assumes pinned supports. This method may be used to estimate the carrying capacity of the arch barrel only and has been

adopted from the method set out by the Military Engineering Experimental Establishment (MEXE), reference [3.6]. The method is based on past experience and is limited to:

- single span arches, of span not greater than 18m;
- arches which are not appreciably deformed or flattened.

The method uses equations as in Appendix 3 from which a provisional allowable axle load for a particular arch may be derived which is then adjusted to allow for:

- shape of arch;
- quality of the material in the arch ring and in the fill;
- dimensions of the arch barrel and presence of any defects;
- condition of the joints.

Engineering judgement still finds its way in this method in the determination of the nature of materials and the state of the structure. Hence, the method is a combination of practical experience backed by a theory of elastic behaviour.

The modification yields the modified axle load which is then multiplied by appropriate axle factors to give the allowable axle loads for all vehicles operating under the Construction and Use (C&U) regulations.

The axle factors cover two situations. The first, is the 'no lift-off' case, which is more usual when all wheels of the vehicle are assumed to be in contact with the road surface at all times. The 'lift-off' case relates to circumstances when an axle of double or triple axled bogie can lose contact, either partially or completely, with the road surface and transfer some of its load to the other axles in the bogie. The derivation of the Axle Factors can be found in reference [3.4].

It should be noted that these allowable axle loads may not represent the strength of the bridge as a whole. This may be affected by the strength of the spandrel walls,

foundations etc. Should the strength of any of these items be assessed as being lower than the barrel strength, then the lowest value should be taken as the strength of the bridge as a whole.

The maximum gross weight of the C&U vehicles which the arch can carry is then found in accordance with reference [3.6]; which is the maximum weight for which both the single and where applicable, the double axle load calculated for the arch are satisfied.

The essential features of this MEXE approach to the assessment of masonry arches are that:

- There is considerable emphasis on the geometrical properties of the bridge; the arch span and the total crown thickness serve to define a provisional value of the axle loading, and the actual shape of the arch is later introduced in the form of modifying factors. One curiosity is that the thickness of the arch ring does not enter directly into the calculations, although it does have a small effect on the value of the material factor;
- the arch is treated, in a late nineteenth-century way, as an elastic redundant structure. A long series of simplifying assumptions is made, but the state of the arch under given loading is evaluated using established elastic techniques;
- the final criterion for the load-carrying capacity of the arch is based upon the attainment of a limiting value of compressive stress.

The whole assessment depends on the values of the thrust and bending moment that have been evaluated at the crown of the arch. The value of the thrust will not be much affected by the various assumptions made in the elastic analysis, but the value of the bending moment is sensitive to these assumptions. On the face of it, therefore, this way of assessing the provisional value of axle load must be regarded with some

suspicion.

However, the criterion of a limiting compressive stress does impose, as it turns out in practice, some uniformity in the assessment.

Further, the MEXE method finds a place for engineering judgement as to the nature of the materials and the state of the structure. However the method is, in the last analysis, an amalgam of practical experience backed by a theory of elastic behaviour which does not really apply to the masonry structure. Also, considering that commercial vehicles and trains have many axles; and railway masonry bridges are multiple arches, this method is of limited value.

3.2.3 Static Loading

This is the conventional civil/structural method of loading a structure or its components by distributed, point or other types of loads and measuring the resulting deflection. Results can be used as a proof test of the structure, i.e. deflections are not in excess of expected values for the applied loading, or bending moments could be computed from the results thereby defining the integrity of the structure.

The load is applied across the width of the bridge at prechosen points, a typical example being at quarter point sections, by hydraulic jacks. The jack loads would be applied through steel beams and timber spreaders to a concrete strip, cast on the road surface. The jacks react against steel beams and the load from each is transmitted to the ground by ground anchors drilled into the rock under the bridge. Figure 3.1 shows the arrangement of a typical loading system.

Displacement transducers are mounted to measure horizontal and vertical displacements at the abutments, quarter points and crown of the arch.

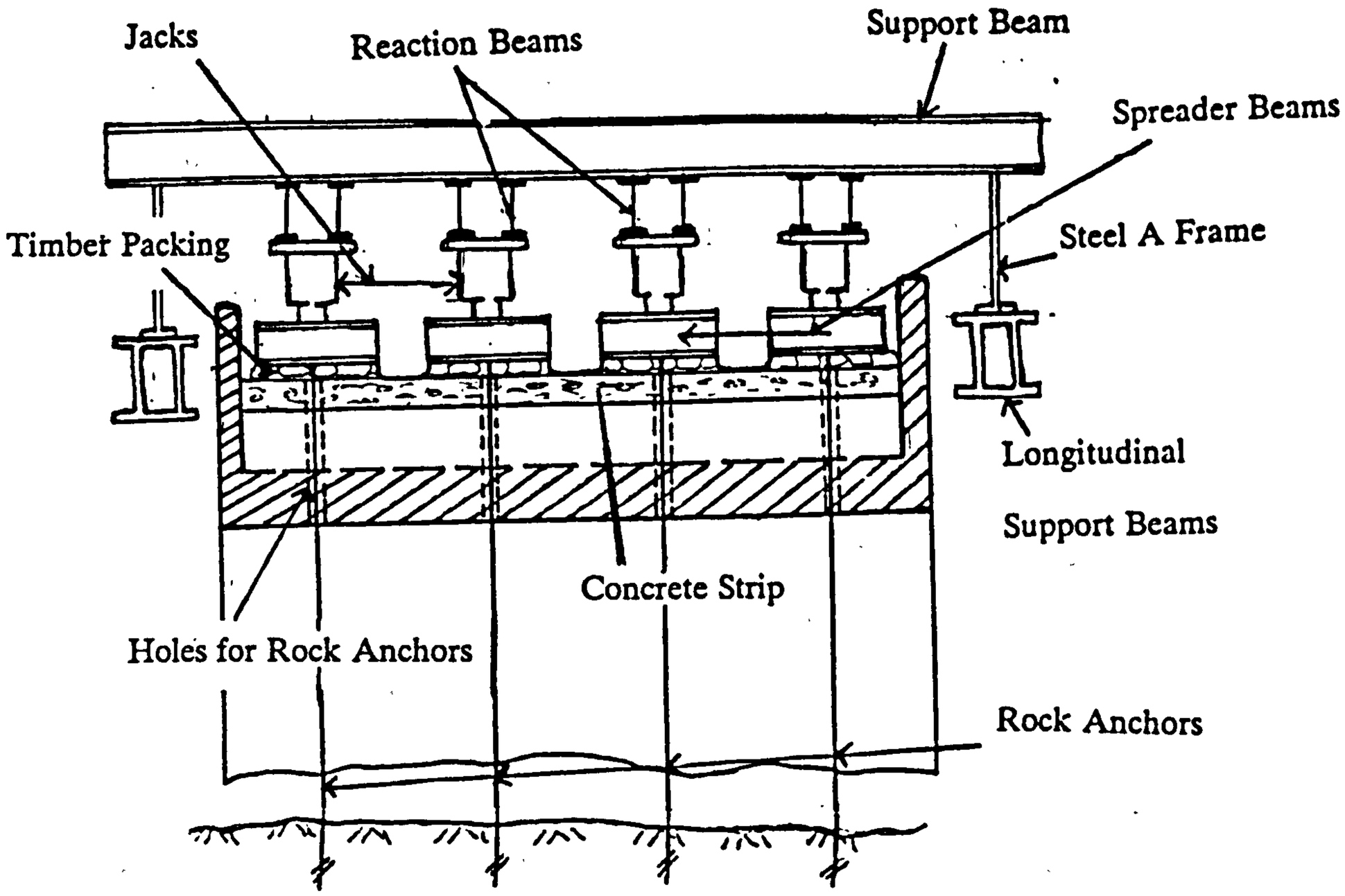


Figure 3.1 Typical Details of Loading Arrangement

However, the method has the following disadvantages:

- expensive because of the heavy machinery in carrying out the tests;
- dangerous as the structure may collapse without warning as a result of rapidly propagating faults such as brittle failure.

The advantages of this type of test are that it gives a failure load and the failure mode of the bridge under test as already outlined in Section 2.4. Also, results from such full scale tests provide information concerning the behaviour of stone masonry arches which would assist in the development of analytical methods for the assessment of the strength of bridges of this type.

The British Transport and Research Laboratory (T.R.R.L.) have embarked on a programme of research work with the object of developing methods for the assessment of

masonry arch bridges. Such tests have been undertaken on behalf of T.R.R.L. by the Civil Engineering Department of Edinburgh University. The test methods used, results obtained, parallel model tests and some comparisons with calculations of two such tests are outlined in references [3.1, 3.2].

3.2.4 Dynamic Signature

The dynamic signature method of nondestructive testing, developed and known by the acronym SHRIMP, reference [3.6], uses a swept frequency vibration of low level to interrogate the structure. The vibrations sweep is injected into the structure at a given point and the responses are monitored by piezoelectric sensors at other points.

Periodic testing by SHRIMP will give an assessment of the structure's integrity (Figures 3.2 and 3.3), and the same method used diagnostically will delineate the fault(s). This is a systems testing method, where the entire structure participates in making up the response at a given point. If a fault such as an internal propagating crack, interrupts the elastic load path from the vibrator to the sensor, the oscillating stress waves making up the signal must travel a different path, thus changing the spectral character of the response.

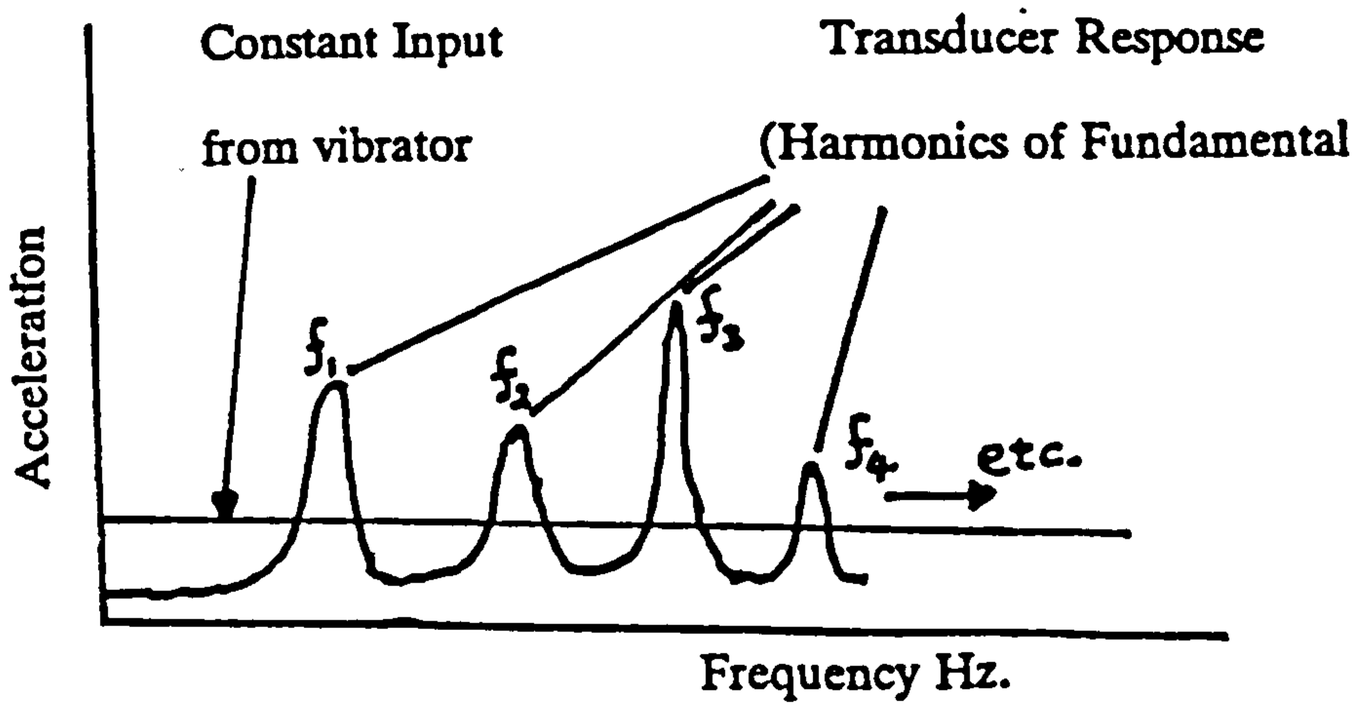


Figure 3.2 Acceleration Signal vs Frequency

SHRIMP testing has also advanced recently by the use of remotely exciting structures

and remotely monitoring the responses. This is accomplished by sonic means and is useful for the interrogation of high voltage ceramic insulators, detecting voids around sewers,etc.

Interpretation of the data is specialised and requires a data bank back-up. This can be achieved by normalising the results from this type of test, for example by storing the amplitudes of the various harmonics (Figure 3.3). Equipment also is specialised and expensive, but companies are available to provide this particular kind of service.

In nondestructive systems testing, a static or dynamic signature is obtained from the entire structure as a response to a disturbing force. An included fault is shown up as a change in that signature, regardless of fault location. Hence the method only signals the presence of a fault, but does not tell where and what type of fault it is, making the method of limited use on its own, and would be of better use with additional back-up information as is provided by a data base.

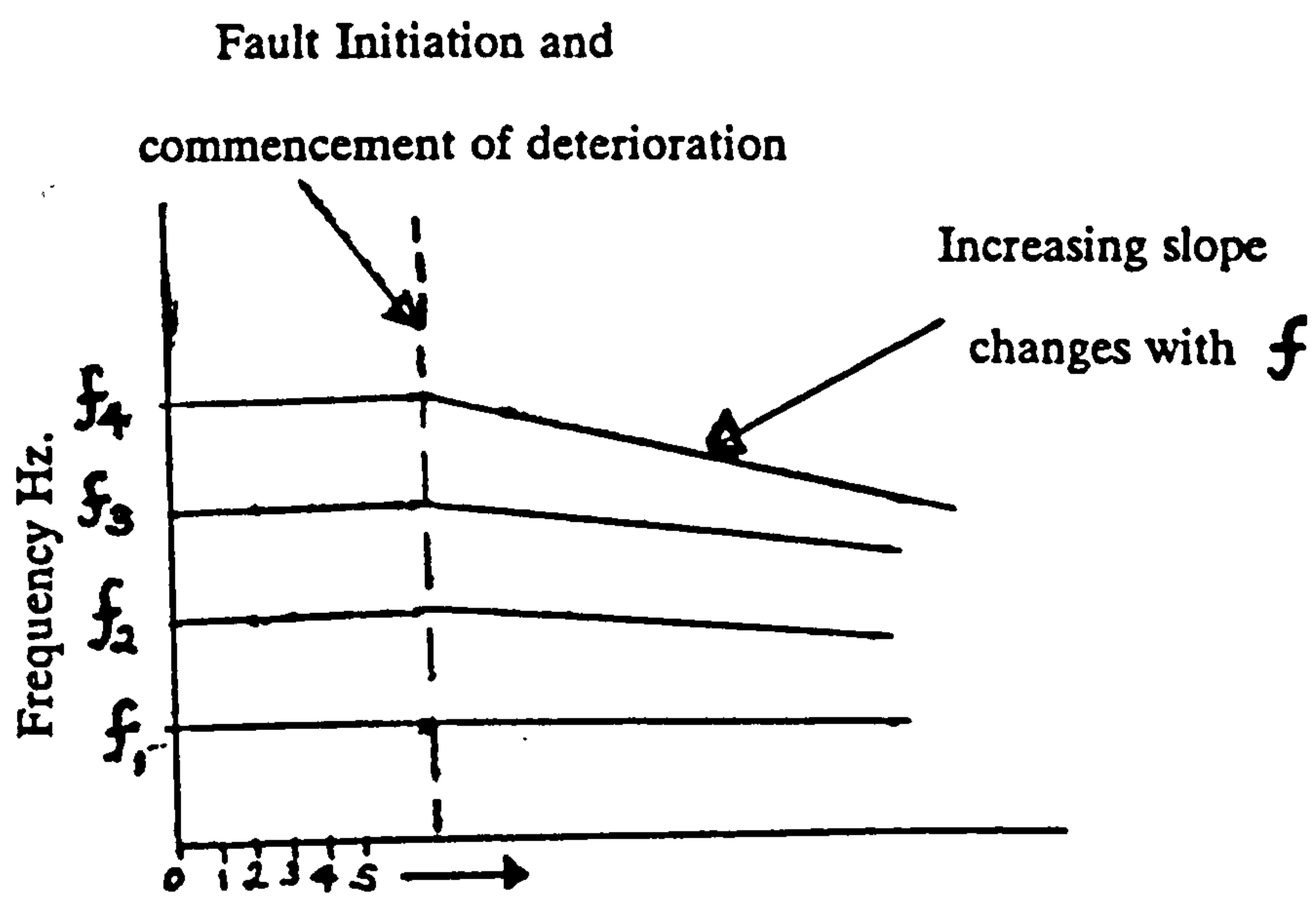


Figure 3.3 Frequency Response As It Changes With Time, Due To Growing Fault.

3.2.5 Sonic Assessment

The sonic assessment technique allows rough determination of the following parameters, viz, thickness of different structural parts, for example, abutment, location of internal changes in sonic properties (these may be associated with cracks or with interfaces between materials of different sonic characteristic), and, by calculation of average velocities of sound at different points on the bridge, the local material strength can be inferred. If the sonic-velocity measurements are made at regularly-spaced grid points over the bridge, a sonic-profile may be obtained (Figure 3.6). This, in turn, can be used to indicate the extent of regions of broadly similar sonic velocity, and, by implication, broadly similar material and mechanical properties, (Whittington,1984). The method yields a crude assessment of the bridge parameters but is, in itself, not definitive. It requires considerably more complementary information about a given bridge before its results can be used with any confidence.

Sonic velocities can be determined by measuring transit times for sonic compression wave propagation between points whose distance apart is known or by indirect reflection, where calibration of the technique is done at two or more coring points and the assumption is made that data from these points may be used for other measurement points on the bridge. The most accurate results are obtained by the straight path method but even here some very broad assumptions must be made regarding the homogeneity of the material through which the sonic wave passes.

The basic principle of the technique entails the propagation of a compression wave through the material of the structure, produced by a blow using a conventional hammer covered with several layers of soft paper, to avoid damaging the brick surface. Two piezo-electric transducers are attached to the two opposite ends of the structure, using water pump grease as an acoustic coupling medium. The transducers are placed opposite each other at the same height above the base of the structure and are con-

nected to a two channel transient recorder.

The time scale and sensitivity of the transient recorder are adjusted such that a relatively light hammer blow, very close to transducer A, (Figure 3.4) would generate a compression wave which would travel between the transducers A and B.

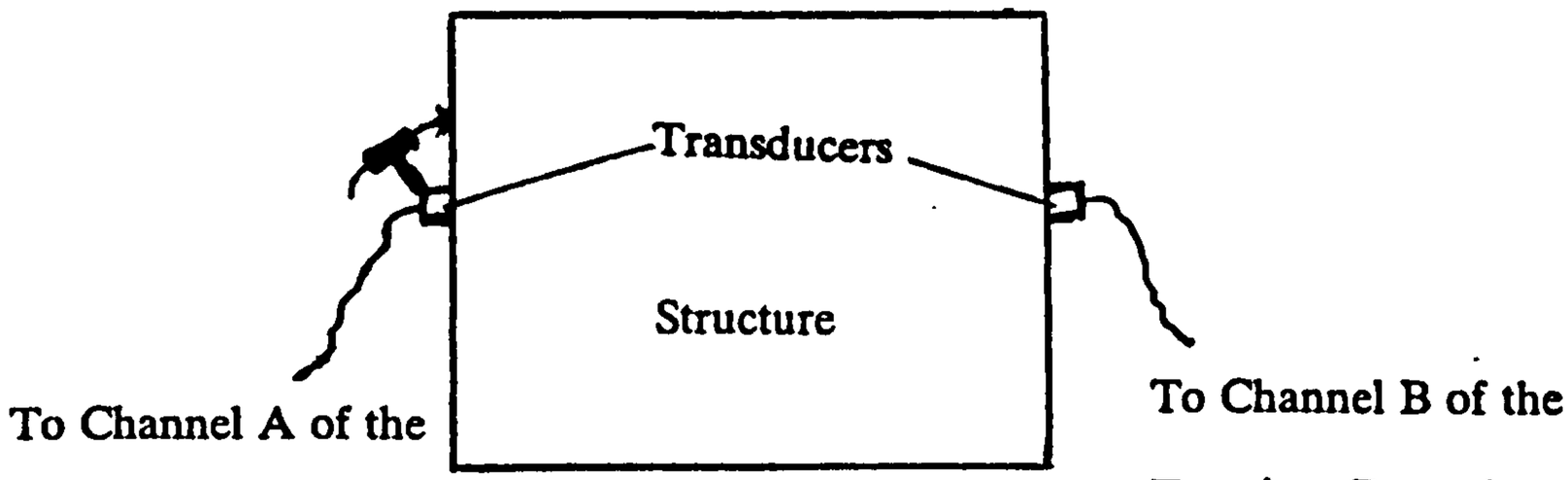


Figure 3.4 Set Up For Transmission Test

Interpretation of Results

Under idealised conditions, when a compression wave propagates through the structure, it will have a shape which can be derived by Fourier's analysis (Figure 3.5).

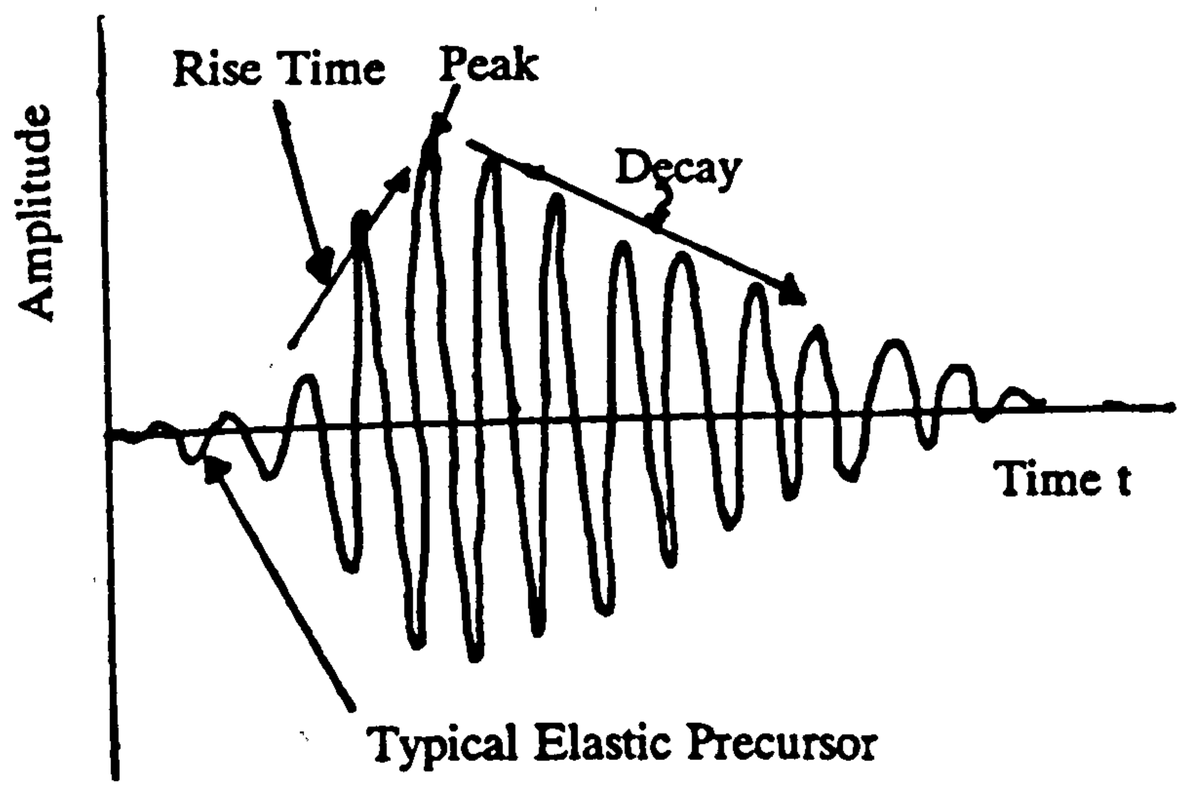


Figure 3.5 A Typical Acoustic Emission Signal

This type of signal with an exponential rise and decay is common in ultrasonic testing. Indeed all the traces obtained do have this exponential rise and decay in common.

However, masonry is a heterogeneous material and additionally masonry is usually cracked. Therefore the traces obtained are more complicated than the idealised case. The complication is due to interference of reflected waves from masonry-mortar joints, any crack or discontinuity and the surface waves generated at the transducer.

The relative proportion of the initial signal transmitted or reflected from a discontinuity, is dependent upon the magnitude, shape and orientation of the discontinuity. For example, a vertically continuous void results in no transmission to the end of the wall as the characteristic impendance of air is much lower than that of brickwork and hence almost total reflection occurs.

In this way sonic profiles may be obtained (Figure 3.6) for use as one element in the assessment process for a given bridge.

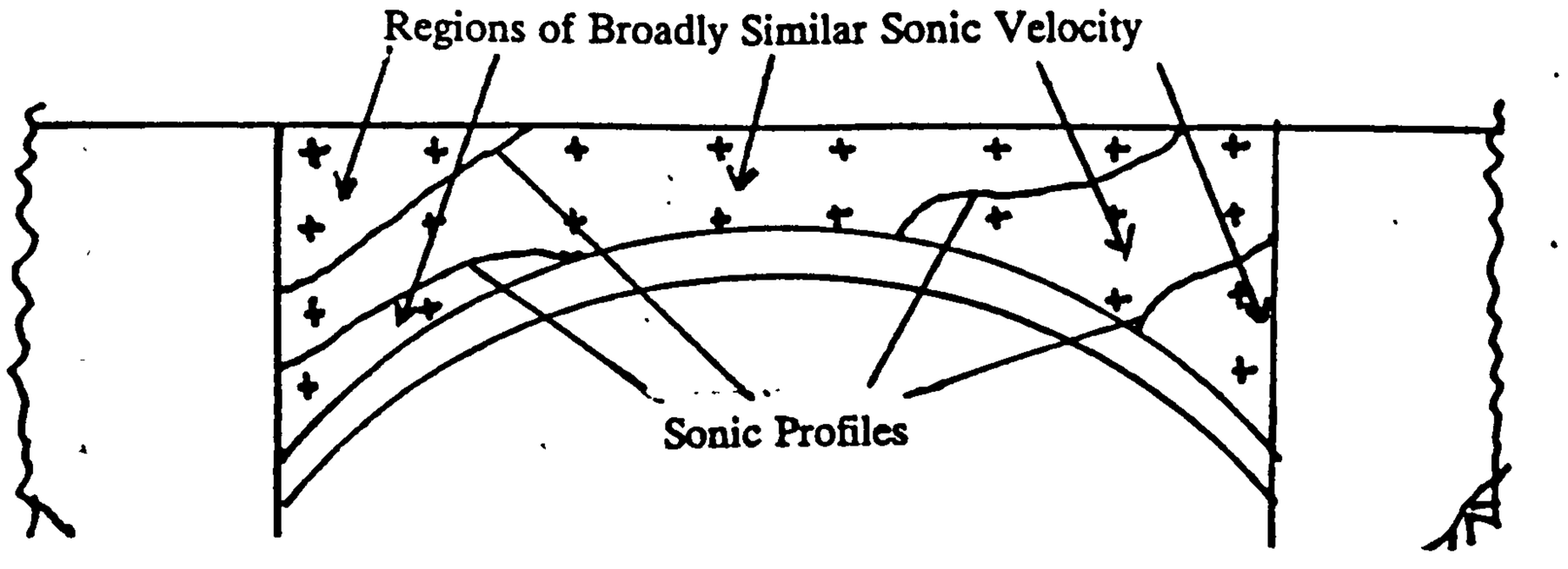


Figure 3.6 Sonic Profiles

Velocity of Propagation

The velocity (v) of propagation of sonic compression waves in any medium is governed by density (ρ) and the modulus of elasticity (E) of the medium, viz

$$v = \sqrt{E/\rho}$$

Because of this dependence on E and ρ, it is possible to infer mechanical properties to materials on the basis of measured sonic velocities. It has been found that the value of

E, for most structural materials, is very sensitive to the mechanical quality of the materials, whereas the density variations are less marked.

Transmission Tests

Here, two transducers are used, one to record the sending end signal and one for the receiving end. The time between the signal being sent and received can be used to determine the propagation velocity of the sonic compression wave. This velocity, in turn, is a measure of the quality of the material between the transducers. Basically good material is characterised by a high sonic transmission velocity and poor quality material by a low velocity although wide variations in measured velocities will exist for the same material.

Typical values of transmission velocities obtained from sonic tests, in metres per second are:

- good brickwork (uncracked) - 3100 ;
- poor brickwork (cracked) - 2500 - 2700;
- uncracked reinforced cavity - 3500;
- cracked reinforced cavity 2700 - 3000;
- structural concrete - 4500;
- loose soil - 300;
- compacted soil - 500;
- stone - 2000 - 7000.

Reflection Tests

Here, a single transducer is used to monitor both the transmitted sonic signal and any reflections which occur. This technique can be used to determine the wall thickness of different parts of the structure such as abutments and barrel and the quality of the

material in the wall. It is used in conjunction with conventional coring, where the core provides a reference for both wall thickness and a datum for material quality assessment. However, since coring just tests the material in the core, using the core results as a reference could be erroneous, hence the method is of limited value.

The sonic testing systems can be calibrated in the following ways.

- the velocity of transmission of signals across the width of the bridge can be calculated directly, since the width of the bridge can be measured;
- the transmission velocity of the materials extracted by conventional coring can be measured directly;
- the apparent thickness of a given wall from sonic testing can be confirmed by coring at the point of sonic testing;
- a data base of the sonic characteristics of different materials can be developed and is used to give reference information on any bridge under test. For example, a relatively low transmission velocity is characteristic of poor quality material since the transmission velocity is a function of Young's modulus for a given material. It must be emphasised that the velocity associated with acceptable quality material will change from masonry structure to masonry structure making the development of a data-base essential for the successful application. This would enable the common features to be stored, then in future, after retesting, changes can be identified.

3.2.6 Tale Tales

This method a means of continuous monitoring of cracks on a structure. Two points are marked at both sides of the crack as shown in Figure 3.7. The separation (d) between these points is measured, and at a later date, the same measurements are made. Any change in the separation of these points can thus be detected. For example,

an increase in the separation indicates that the crack has widened.

Figure 3.7 Tale Tales

3.3 TECHNIQUES FOR REPAIR AND MAINTENANCE OF MASONRY BRIDGES

A distinction can be made between ordinary and special maintenance: the former includes simple operations such as cleaning of drainpipes, filling small cracks and replacing individual bricks or stones which may have fallen. Operations in Sections 3.3.1 - 3.3.5 fall into the category of special maintenance. Where it is necessary to carry out a major restoration operation, these are better classified as repair rather than as maintenance. The remainder of this section describes such repairs.

Repair solutions can differ markedly depending upon the extent and the type of damage. For example, if a span of a footbridge is entirely removed by vehicle collision, a new span would be provided, Photograph 3.I. Repairs to a structure which is intact and usable should be carefully detailed so that they are effective and can be executed safely and with the minimum of disturbance to users of either the structure or the facility beneath. Possible methods of repair for different types of structures are described in the following sections with comments on their suitability in particular circumstances.

3.3.1 Filling of Cavities or Cracks by Injection

It is advisable to limit this operation to cracks no larger than 5mm. Larger ones should be dealt with by means of partial replacement of the areas concerned.

Masonry can also be injected with cement products; in such cases care should be taken

that the grouts have a low level of sedimentation and are non-shrinking. This can be ensured by means of the same technique as those used for filling post tensioning ducts (high-speed mixers to prepare the grouts, which should be of the colloidal type to permit low water/cement ratios).

Care should be taken to avoid obvious mistakes such as injecting and thus blocking the drainage systems; a rigid bond between the masonry and the support should also be avoided, insofar as this could prevent the arch from moving freely in response to changes in temperature.

It should be kept in mind that simply filling in cracks in the structure of an arch, without attempting to tackle the causes of the cracking, is not only useless, but may well be counter-productive. For example, the relative settlement of the piers due to overloads on the road beneath the structure can produce cracks in the crown; these remain active and, if plugged, will re-form until such time as the causes cease, the settlement reaches its limit, or, better, the pier is reinforced.

Cracks in the arch can also give rise to more extensive breakdown processes, not directly related to the original causes. These, for example, can result in movements between the stones or bricks which in turn lead to irregular distribution of pressures in the arch. In such cases it is necessary to have to resort to the suitable maintenance interventions described below.

Finally, it should be remembered that the structure and functioning of the arch are such that when horizontal cracks develop in the crown, wedge-shaped in the direction of the thickness starting from the intrados, it can be presumed that a similar cracking process is taking place in the springing; this is identical in every way except that it starts from the extrados, and thus appears to be less serious and more circumscribed, but is of similar size inside the body of the vault. Without going into a discussion of the origin of such crack patterns, which may be due either to overloads or to thermal

phenomena, decides to fill them, such cracks in the springing will require much deeper injections than might seem necessary at first sight. In particular, there is a proportionately greater risk that the injection material will disperse into secondary cracks in the masonry.

3.3.2 Improvements of Drainage and Water Evacuation Systems

Copings of waterproof materials were not common in the past, and consequently many surviving structures not so equipped are characterised by substantial and undesirable internal circulation of water.

In other cases the longitudinal slopes and crossfalls on the extrados are not sufficient to drain off the rainwater. Depending upon circumstances, it is thus necessary to equip the structure with gutters and pipes which will carry off the water without discharging it on the piers or supports, or with efficient outlets which correspond with the drainage system. if such already exists. Where such systems are lacking, a series of collectors can be installed at the points most susceptible to penetration by water; they may also serve to collect water circulating within the structure, in the absence of more thorough measures.

It is evident that the zone to be treated must be carefully selected, possibly on the basis of suitable tests; similar care should be taken in choosing the diameter of drainpipes, which may possibly be slotted or perforated and protected with a covering of non-woven fabric so as to limit the transport and loss of fine aggregates.

3.3.3 Waterproofing of the Extrados

The waterproofing layer should closely follow the profile of the arch, and be laid over a well-smoothed support layer. Caping layers of cement mortar are not recommended insofar as they may tear the waterproofing material. The most suitable materials to

employ are asphalt and bituminous sheeting.

3.3.4 Repointing and Restoration of Masonry

This operation represents the most suitable way to restore the original appearance of the structure.

It is sometimes not necessary to replace the stones or bricks themselves, but only to replace the mortar bedding. In such cases one starts by carefully chiselling away all the old binder or washing it out with high pressure jets of water, possibly removing temporarily certain of the masonry elements. The resulting cavities should be cleaned, smoothed and dampened so as to receive the fresh mortar, which must be carefully inserted and compacted.

3.3.5 Installation of Steel Cross Ties

These serve mainly to prevent the breaking away and bulging of the side walls with respect to the structure of the arch itself, but also to prevent the spread of possible longitudinal cracks.

The cross ties can be very useful, but may also lead to unforeseen redistribution of forces; they should be employed with care, in suitable numbers and be of limited diameter. Close attention must be given in boring the holes for the ties: the methods will vary according to whether it is necessary to bore through the supports or the arch structure itself. The tie rods are protected with scaled sheaths; the bolts and anchor plates, which must be of sufficient size to prevent the concentration of stresses on the masonry facings, are provided with anti-corrosive protections. Photographs 3.II and 3.III show this type of maintenance.

Photograph 3.I Replacement of Span With Concrete Deck

Photograph 3.II Replacement of Old Stonewokk With New Stonework

Photograph 3.III Installation of Steel Cross Ties

Photograph 3.IV Installation of Steel Cross Ties

Photograph 3.V Replacement of Coping

Photograph 3.VI Ties in Coping

3.3.6 Installation of Steel Ties

These serve mainly to increase the carrying capacity of the arch or to remedy a defect. A typical case in which they are employed is when a system of cracks develops in the crown and in the springings foreshadowing a yielding under load. They can also be utilised, however, in cases where the intermediate piers are out-of-plumb due to excess loading, but sometimes without the appearance of cracking in the crown. In this case the cracks in the springings will be longer and more extensive than usual, the breaks in the springing opposite the out-of-plumb pier being especially so.

It is advisable to combine the installations of the tie rods with injections into the masonry. In general, the injections are performed first, and then the tie rods are put under tension and the mortar allowed to set. The filling material for the sheaths may be the same as that used for the injections.

3.3.7 Repairs to Abutments and Piers

A system of small diameter bored piling has been shown to be an extremely useful means of providing extra support needed to limit settlement or where additional loading is anticipated. In order to provide continuity the piles are bored through and cast into the existing abutment. Where the abutment itself is weak it may require grouting or stitching together by some means, as example being the system shown in Figure 3.8.

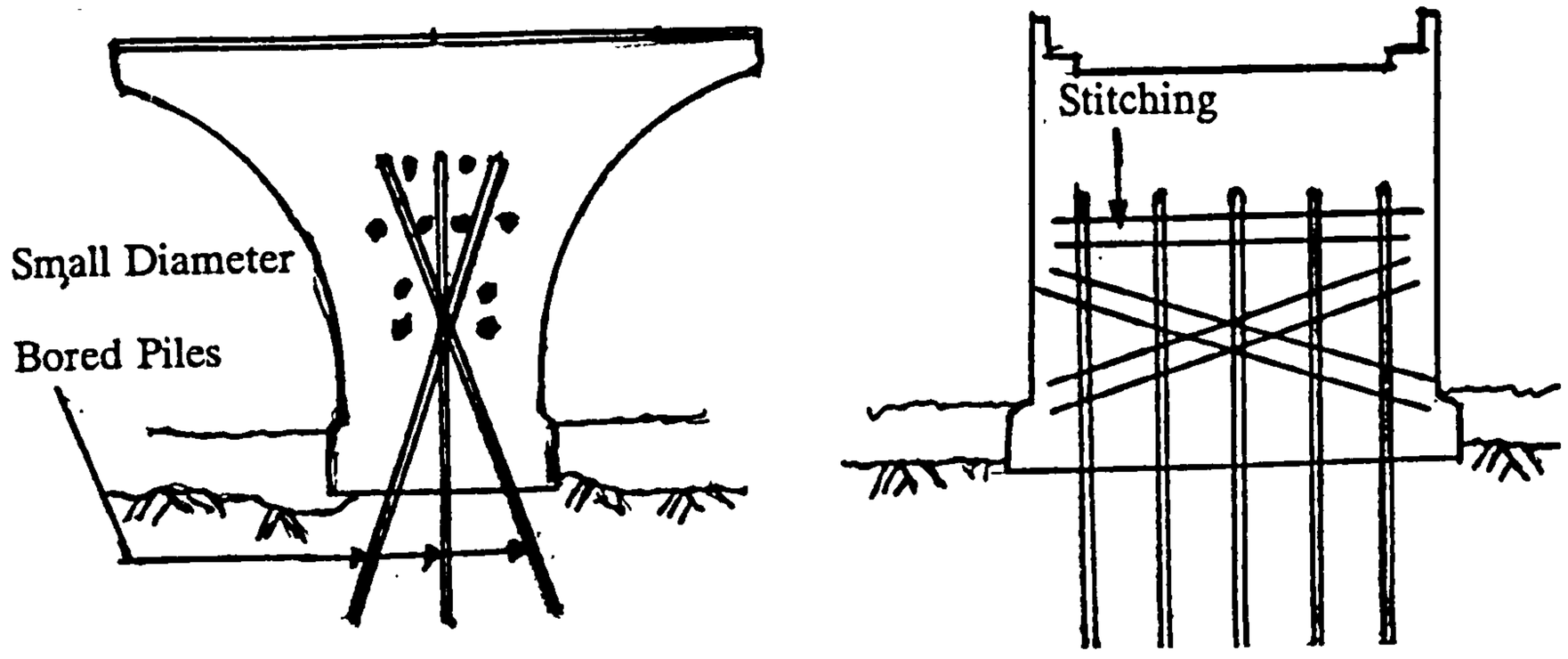


Figure 3.8 Small Diameter Bored Piling and Stiching

Many arch bridges were built on very shallow foundations. This leads to frequent undercutting due to scour and if underpinning is required it is prudent to build a concrete apron or invert slabs around the abutments or pier in order to protect the toe of the masonry, Figure 3.9.

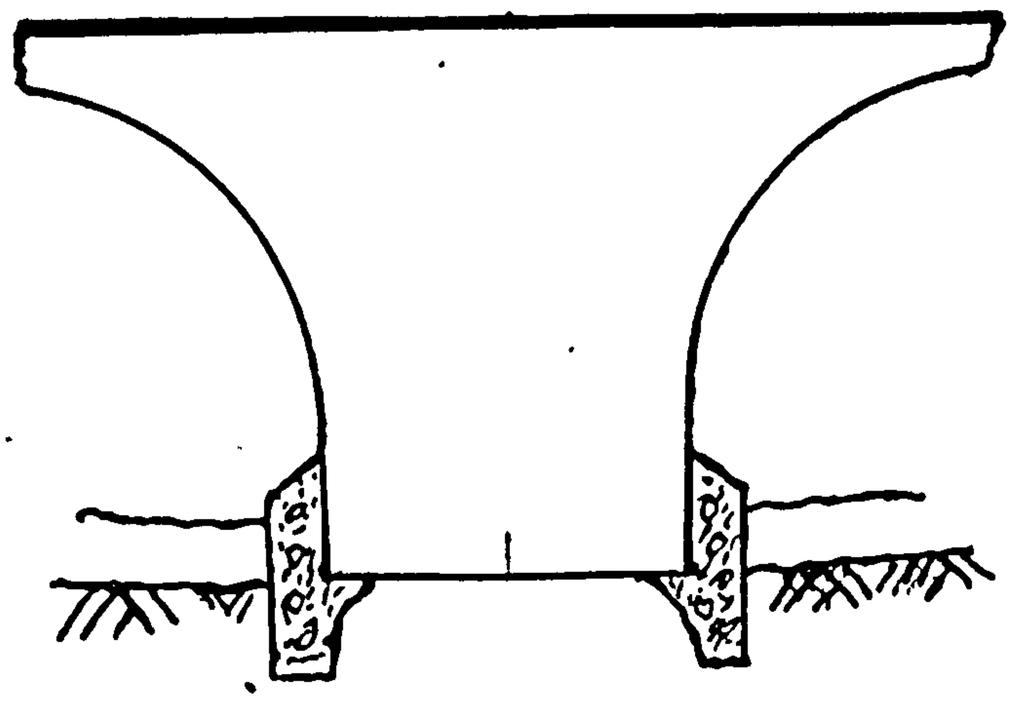


Figure 3.9 Use of Concrete Apron

3.3.8 Repairs to Arch Barrels

The most common means of strengthening an arch barrel is to cover it with reinforced concrete saddle or relieving arch. The advantage of this method is that it not only strengthens the arch but also improves load distribution and ties together any cracked sections. When using this method care must be taken to ensure that the thrust is transmitted to the abutment and that the abutment is capable of carrying the additional load, Figure 3.10.

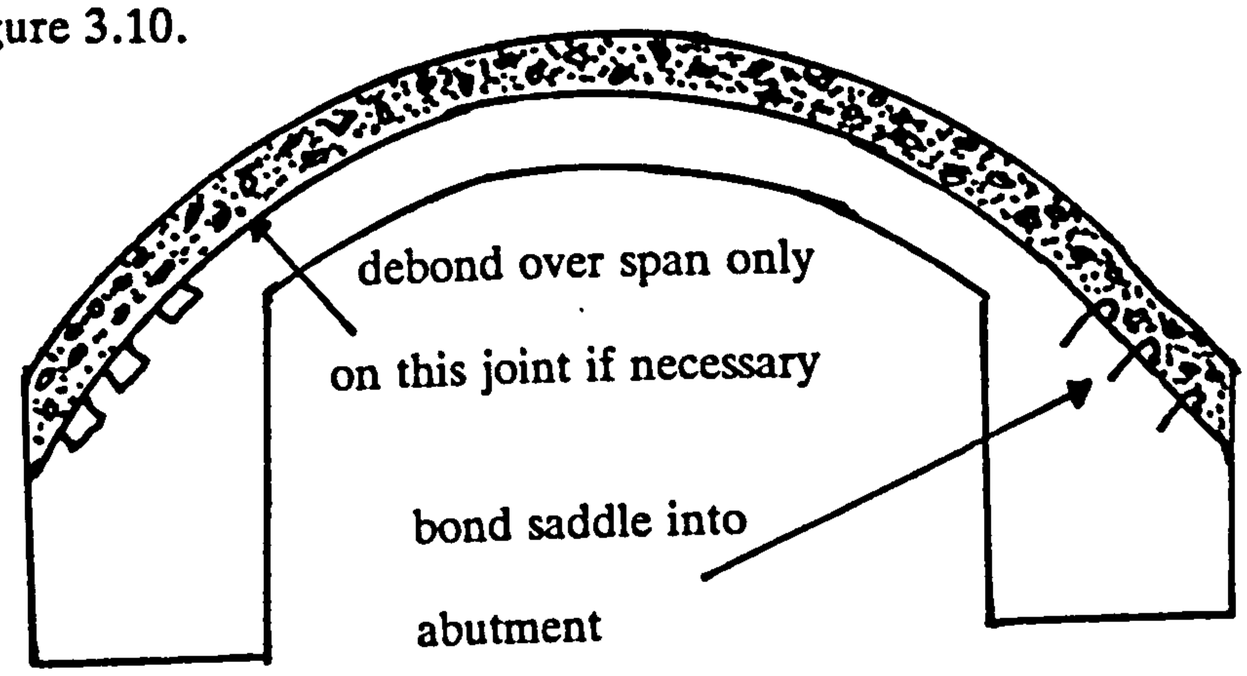


Figure 3.10 Concrete Saddle

It is usual to cast the saddle directly onto the existing extrados thus ensuring composite action. Where no extra stress must be carried by the existing arch then a smooth debonding layer must be introduced. To reduce induced shrinkage stresses the saddle should be thoroughly cured and consideration given to casting segmentally.

If extra thrust cannot be accepted by the abutments then a concrete slab may be built taking the necessary support from the abutments, Figure 3.11.

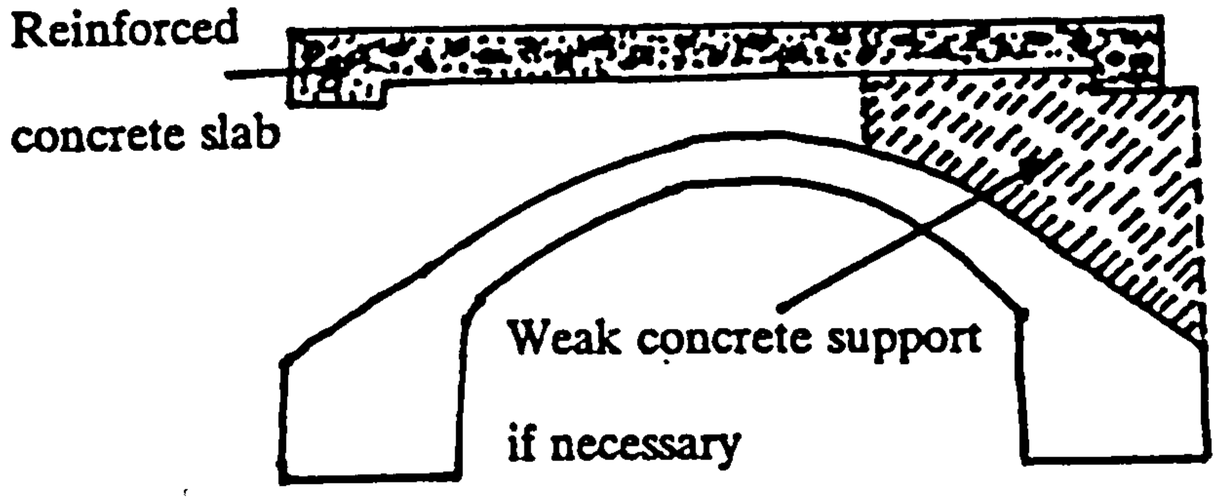


Figure 3.11 Concrete Relieving Slab

Where there is a large depth of fill or where the headroom beneath the bridge is not critical and appearance is not important, it is often economic to place a relieving arch underneath. This may be conveniently provided by sprayed concrete techniques or by placing a corrugated metal or glass reinforced liner within the arch and pumping concrete into the gap between the liner and the existing intrados, Figure 3.12. As a temporary measure during the passage of a mining wave, steel colliery arches may be used, supported by walings bolted to the abutments, Figure 3.13; bent inverted T or I rolled steel beams may also be used to provide support for the arch.

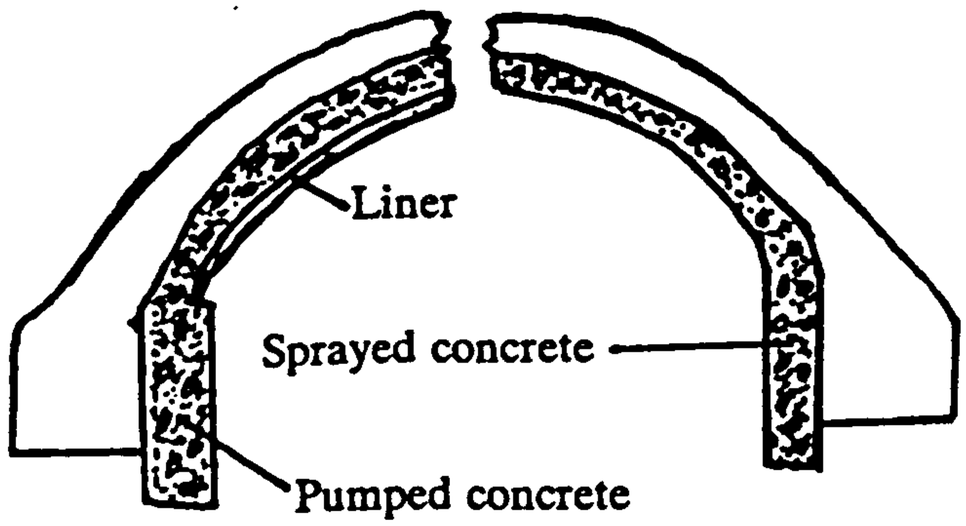


Figure 3.12 Strengthening From Underneath the Arch

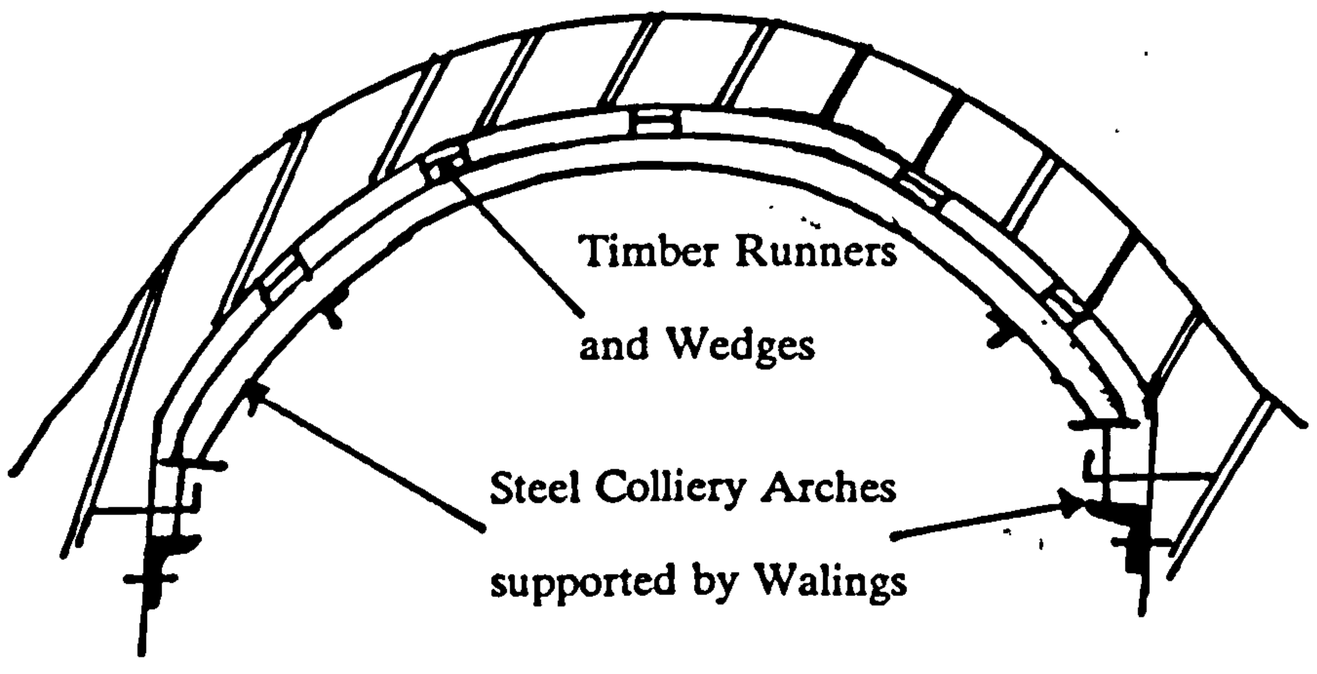


Figure 3.13 Use of Colliery Arches

3.3.9 Repair of Spandrel Walls

The traditional means of repairing walls that were deforming, tilting or sliding off the barrel was to tie both walls together with rods and large spreader plates on the outside of the bridge as outlined in Section 3.3.5. This is unsightly, but has the advantage that it can be carried out without disrupting traffic. Another solution is to expose the walls and backfill them with concrete. If a barrel is being saddled, this is always the most

appropriate method. Alternatively, consideration can be given to the use of needling through the spandrel walls.

3.3.10 Repair to Road Surfacing

Surfacing must be kept in good repair as irregularities cause increased impact loading. Pot holes, lack of camber and cracks allow entry of water. Particular care should be taken to ensure that service trenches are properly backfilled and the surfacing released.

3.4 OVERVIEW

From the outline of the methods of assessment it is evident that Engineering judgement plays a major role in the assessment process while itself heavily relying upon information thereby making it very essential that all available information pertaining to a structure such as soils data and past inspection records be collected and made available. This would facilitate in determining what future information should be obtained from inspection and which components of the bridge require special attention.

The following chapter gives an overview of data bases and data base management systems.

REFERENCES AND BIBLIOGRAPHY

[3.1] Henry, A. W., Davies, S.R. and Royles R., "Test on Stone Masonry Arch at Bridgemill, Girvan", Department of Transport and Road Research Laboratory, Crowthorne, Contractor Report 7, 1985, pp 2 - 6.

[3.2] Henry, A. W., Davies, S.R. and Royles R., "Testing of Masonry Arch Bridge

at Bargower, Stratclyde", University of Edinburgh, C.E.R.T.I., Report to T.R.R.L., 1985, pp 1 - 9.

[3.3] Heyman J., "The Estimation of the Strength of masonry arches", Proceedings of Institution of Civil Engineers, 1969, p. 921.

[3.4] Organisation For Economic Co-Operation And Development (OECD), "Bridge Inspection", A Report Prepared by an OECD Research Group, July 1976, pp 14 - 36.

[3.5] Pippard, A.J.S., "The Appropriate Estimation of Safe Loads on Masonry bridges", Civil Engineer in War, Volume 1, London, The Institution of Civil Engineers, 1948, p 365.

[3.6] Savage R.J., and Hewlett P.C., "SHRIMP", Annual Conference on NDT, British Institute of NDT and Society of X-ray Technology, University of Aston, Birmingham, Sept. 1978, pp 82 - 93.

[3.7] Scottish Development Department, "The Assessment of Highway Bridges and Structures", Technical Memorandum(Bridges) SB3/84, Printed and Published by the Scottish Development Department,1984.

[3.8] Scottish Development Department, "The Assessment of Highway Bridges and Structures", Annex I to Technical Memorandum(Bridges) SB3/84, Printed and Published by the Scottish Development Department,1984.

[3.9] Structural Faults 83, Proceedings of the International Conference on Structural Faults, University of Edinburgh, 22-24 March, 1983, Edited by Forde M.C., Whittington H.W., and Whyte I.L.

[3.10] Whittington H.W., "Sonic Testing of Civil Engineering Sub- and Super-Structures", Proceedings of IEEE, Symposium, Ultrasonics, Dallas, U.S.A., 1984.

CHAPTER 4

DATA BASES AND DATA BASE MANAGEMENT SYSTEMS

4.0 BACKGROUND

The successful everyday running of any enterprise relies on the resources at its disposal.

The nature of these resources varies according to the nature of the enterprise. More commonly, these resources are taken to be such items as the equipment and machinery people, money and materials, buildings etc. Although abstract and less tangible, data are indeed a resource to any enterprise.

Systems that provide this data vary from the most primitive systems based on pieces of paper to systems which make use of modern computer hardware and software. Within this wide spectrum of systems is the data base system which is the latest means of data storage.

Computer data storage dates back to the 1880s reference [4.4] when Dr Herman Hollerith of The U.S. Bureau of Census invented punched cards which remained the leading means of data storage for the next 60 years. By the mid 1960's, the idea of Management Information System (MIS) gained momentum but was short lived because of its lack of application-program and language independence, that is, intergration. This however, highlighted the need for greater intergration and in 1965 General Electric (now owned by Honeywell) produced the Intergrated Data Store (IDS) which was a forerunner of the modern Data Base System. MIS packages based on intergrated files for major systems soon came to the scene after IDS, but soon the problem of lack of coordination between the files of the major systems became apparent.

The essentiality of a data base containing a generalised intergrated collection for all systems of an organisation serving all its application programs was realised from the problems outlined above. This led to the concept of a data base which was application program and language independent. This concept of a data base took off in the early 1970s. A number of data Base Systems based on this concept appeared on the market giving variable performance and hardly any compatibility.

In the late 1960's Codasyl developed an interest in data bases and his work led to the development of standard data base model, the Codasyl model reference [4.1]. At the same time IBM Research Laboratories produced two models, the Relational Data Base Model and the Data Independence Accessing Model.

4.1 DATA PROCESSING

4.1.1 The Traditional Approach

Data processing systems, in the traditional approach were mostly systems designed in isolation, that is, independent of other related subsystems. A typical system would consist of a large number of small programs with many files each containing fragmented information. Thus, operating systems were principally batch-oriented with a complex application split up into a series of jobs, each operating on its own particular files, to be performed in some predefined logical manner. Unfortunately, inherent in this application oriented file structure approach were several disadvantages in the following major areas:

- **Redundancy**

  To carry out particular tasks, in the traditional approach, it is necessary to create duplicate copies of files, as for instance one application may need the same data as another but sorted in a different manner. Also, frequently-run applications

often use independently maintained subsetted files for improved overall system efficiency and unless considerable care is exercised in the management of these copies, changes made to the originals would be reflected in the copies with consequent data inconsistencies arising.

- **Sharing**

The sharing of data between different application systems is in most cases a difficult task usually requiring the development of an independent system to reconcile the various implementation and structure differences.

- **Standards**

Application oriented files may apply different rules for the representation and validation of the same data items. However these rules are contained in the procedural logic of the application program, hence, the same item of data in one part of the system may be treated differently in another.

- **Data Independence**

File descriptions are directly related to an application, implying that any change in the physical organisation of a file or any logical modification to the file, e.g. the addition of a new data item, would necessitate source changes to all the application programs using that particular file. In other words, application programs are data dependent.

- **Access Control**

Any form of detailed privacy constraints are difficult to achieve as programs would either be completely denied access to a file or wholly permitted access to the file.

■ **Different Data Views**

Providing different views of the data for different applications requires that the data files be sorted or shuffled in some way. This often destroys one of the advantages claimed for the traditional approach, namely that application-oriented file structures provide the most efficient solution to a data management problem.

4.1.2 The Data Base Approach

Developments in disk technology made the storage of large volumes of data in directly accessibly form an economically viable proposition, leading to a development of interest in the data base approach to data management problems. This approach overcame the problems associated with the application-oriented approach by providing a data base management system which:

- controlled redundancy - a data base reduces data duplication and provides consistent and up-to-date data.
- facilitated data-sharing - a data base permits more than one program to access the data base at the same time, thus maximising resource utilisation.
- applied standards uniformly and consistently.
- enabled sophisticated access controls to be applied - in a data base, privacy and integrity of data can be controlled.
- provided independence of data and program - this is the prime advantage of a data base. Both the data base and the program can be altered independently of each other. This saves both time and money spent on modifying them to retain consistency.
- supported different views of the data base.
- provided rich data structuring capabilities - a data base containing a mass

of data items will be of little use unless the data is structured in a meaningful way. Therefore, a DBMS must provide data structuring facilities which are capable of expressing the often complex relationships which may exist between the data items (e.g. which products have ordered by which customers in what quantities and which suppliers can supply them?). Furthermore, the DBMS must enable quick access to the data to satisfy the various needs of users.

- provided ease in systems design - a systems designer in a data base environment is not concerned with extensive file design, data duplication, data inconsistency, maintenance and backup facilities of a conventional system. In a data base, data exists in a form suitable for all applications, hence the designer has only to select what he needs thereby making systems design easier.

- eased programming - the programming task is significantly reduced since the programmer is relieved of the details of file processing, file updates and extensive sorting associated with file systems.

- supported multiple host languages - language independence is another virtue of a data base system. A data base can support a number of host languages thereby offering the user the choice of the language most suited for a particular application.

- enabled the development of evolutionary systems - the user of a data base can build up the data base gradually, learning from experience. Facilities for reorganising the data base to optimise the overall performance are provided. Because of data independence, as already mentioned above, the changes in the data base would not affect the application programs.

The disadvantage of a data base is the cost. It is expensive to install, run and maintain.

4.2 DATA BASE MANAGEMENT ARCHITECTURE

Figure 4.1 shows a generalised structure of a data base management system. In this architecture, data is perceived in a number of different views, as follows:

- the schema (or logical data model) is a description of all the data and its structure and will include any access or integrity constraints which apply to the data;
- the sub-schema (or logical data sub-model) which describes the local views of the data base required by the application program; program; Sub-schemas normally provide a restricted view of the data which the applications programs require thereby providing a privacy mechanism;
- the storage data model provides the system with a perception of how data is to be stored and accessed. This model defines which records are to be stored and how they are to be accessed;
- the device data model is the view of the actual physical data, the data that can be seen if a dump of the data base were to be printed;
- the data base control system (DBCS) is a software module which will access the data base in response to a data manipulation language (DML) commands.

Between each of the first four components mentioned above, together with the application programs, are definitions of the way records are associated with one another, that is, mappings. The DBCS is responsible for effecting these mappings. At each level of data model, a data description language enables the particular model required to be clearly expressed.

In order to obtain an idea of how the generalised architecture works in practise, consider the following example:

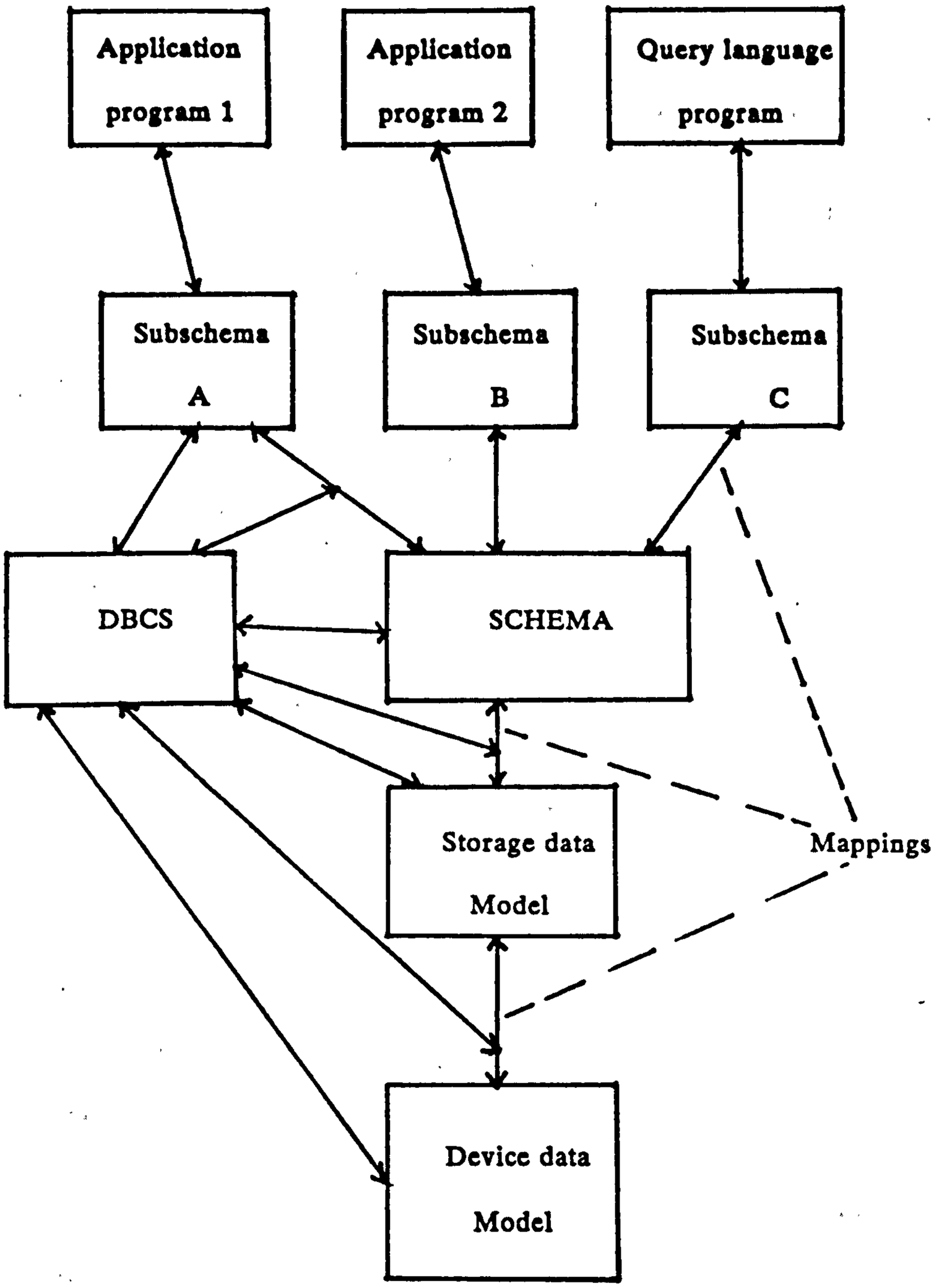


Figure 4.1 Data Base Architecture

Supposing an application program 1 (Figure 4.1) issued a DML command to store a new record in the data base. The DBCS would consult the application program's sub-

schema, the schema, the storage data model and the mappings associated with them in order to find out how the record should be stored. The DBCS could then command the operating system to store the record in the data base. Finally, a message would be sent to the application program to say whether or not the operation was successful.

This sort of generalised architecture is designed to meet the objectives of the data base approach, as outlined in Section 4.1.2. The user of schema which provides rich data-structuring capabilities, enables controlled redundancy to be applied to data, data sharing, and the enforcement of uniform standards. Use of the schema concept also allows sophisticated access controls to be applied. The various models provided for data at different levels facilitates data independence, and the sub-schema concept provides different views of the data base.

4.3 DATA BASE MODELS

Data bases can basically be divided into two groups, namely, the formatted and the relational. Classification of data base management systems tends to be based upon the way in which their schemas (or logical data models) enable data to be structured. From formatted data bases two major classifications have evolved, namely the Hierarchical model and the Network model, while from the relational data bases has evolved the Relational Model.

In formatted data bases a variety of data structures is used to represent relationships. These structures are complex and usually involve pointers to relate records logically. A range of data structures from simple to complex is supported to facilitate the required access paths, which must be specified explicitly. Hence, a data structure can support only a limited number of predefined access paths and, unless an access path is specially provided, a given unit of data cannot be accessed.

On the other hand, in relational data bases, all data structures are reduced to two

dimensional tables of specified characteristics, which are mathematically known as relations. Access is provided to each item directly through algebraic operations on relations; i.e. access is universal. The Relational model is discussed in chapter 5 while the rest of this chapter concentrates on the formatted models.

4.3.1 Hierarchical Model

A hierarchical file is a file with a tree structure relationships between the records. A tree is composed of a hierarchy of elements, called nodes. A branch may become a node thereby generating further branches, thus giving rise to successive levels of hierarchy. A node is known as the parent while branches from the node are known as children. A tree is identified by its top most node, the root commonly called the root record.

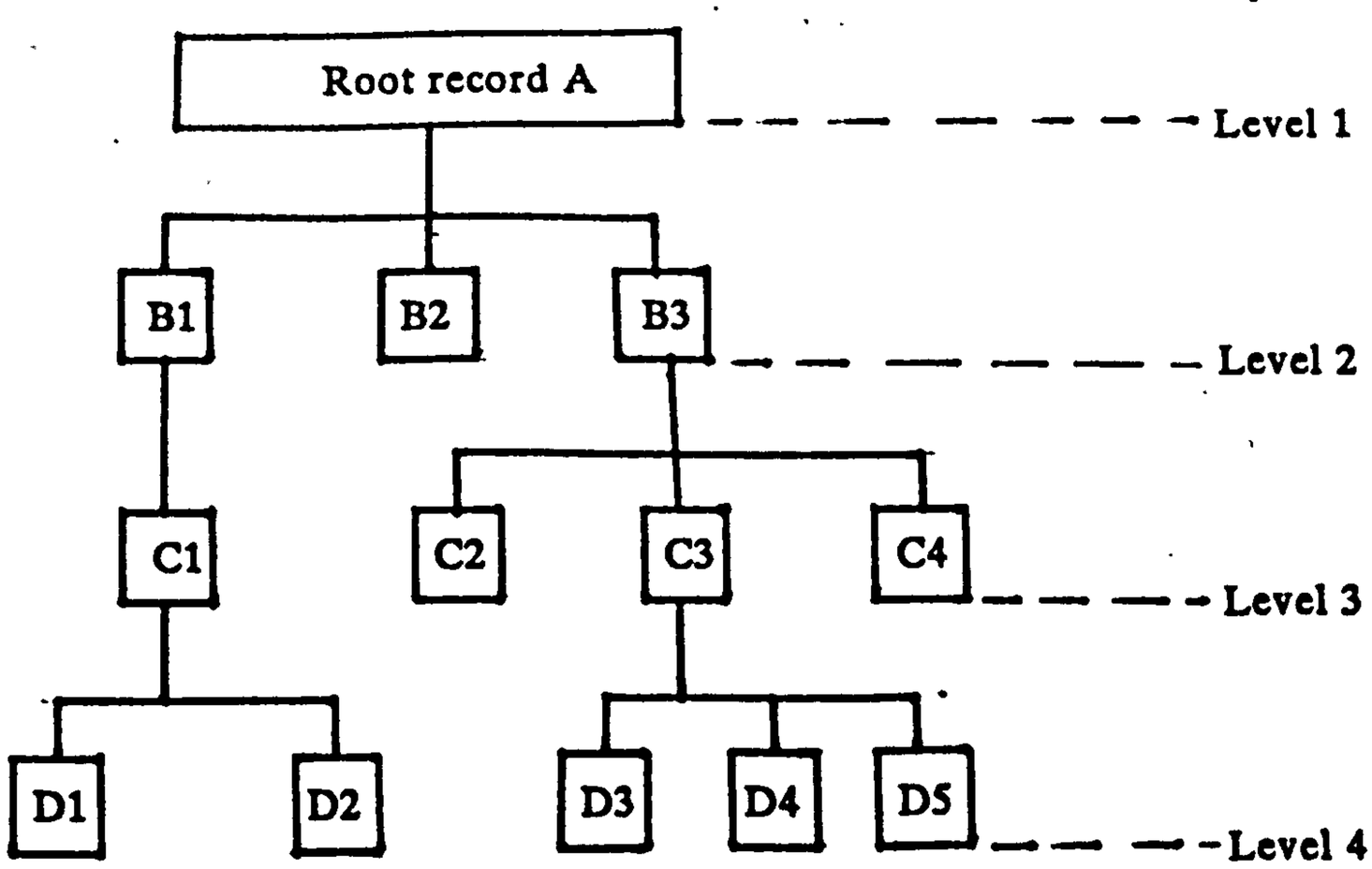


Figure 4.2 Tree Structure with Four Levels of Hierarchy

Figure 4.2 is an example of a tree structure. The records in the tree could be the components and subcomponents of a product A which is made up of part numbers B1, B2 and B3. Part number B1 is made up of part number C1, which is composed of D1 and

D2. Likewise, part number B3 consists of C2, C3 and C4, C3 itself being constructed from D3, D4 and D5. The example illustrates a tree showing four levels of hierarchy.

Returning to the concept of hierarchical file, if we replace the term 'element' in the definition of a tree by 'record', then we have the definition of a hierarchical file. Figure 4.3 illustrates a hierarchical file structure with two levels.

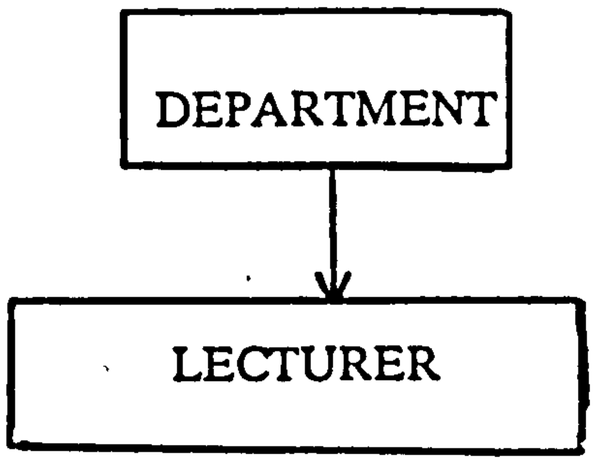


Figure 4.3 Hierarchy

4.3.2 Network Model

Using the description of a tree structure, if the child in a data relationship has more than one parent, then the relationship cannot be strictly hierarchical. In these circumstances, the structure is described as a network. Figure 4.4 is an example of a network structure.

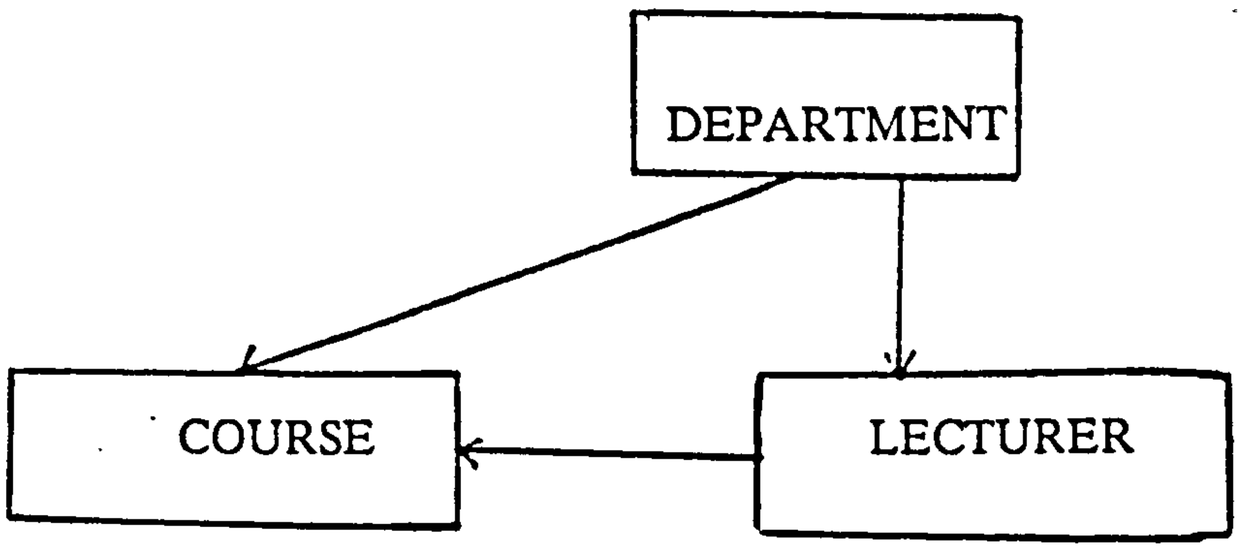


Figure 4.4 Network Structure

4.4 DATA BASE NAVIGATION IN FORMATTED SYSTEMS

The way in which applications access and update a data base varies according to the data base model employed. Associated with each model is a Data Manipulation Language (DML) whose syntax and semantics vary from implementation to implementation. However, in general terms, the language constructs employed are similar in that they reflect the underlying structures of the particular model in use.

In formatted systems, it is essential to be able to navigate the data base and then using these occurrences as a base from which to traverse the various pre-defined relationships that connect record occurrences together. To demonstrate this, consider the hierarchical structure (Figure 4.3), to be searched for all lecturers belonging to the Engineering department. The following data manipulation statements may be deployed:

```
    FIND DEPARTMENT RECORD WHERE
      DEPT_NAME='ENGINEERING'
5   FIND NEXT LECTURER RECORD
      VIA LECTURERS_IN_DEPT SET
    IF [END_OF_SET] STOP
    PRINT LECTURER_NAME
    GO TO 5
```

The first statement finds the particular record occurrence in the data base of the DEPARTMENT record which has the value 'ENGINEERING'. Having found this record occurrence, the next statement traverses those LECTURER records connected to the Engineering department record. (The relationships in formatted data bases are often referred to as sets; hence the use of the term SET in conjunction with the relationship name LECTURERS_IN_DEPT). The third statement simply tests to find out if the LECTURER records connected to the Engineering department record have all been processed and, if so, to stop the application.

Data manipulation languages for formatted systems contain a variety of facilities for the modification, deletion, and establishment of relationships amongst record occurrences. These languages are normally accessed through a high level programing language such as FORTRAN or COBOL. The statements are either directly recognised by the high level language compiler, or, more commonly, the source code of the application containing data manipulation statements is pre-processed by special purpose program which converts the data manipulation statements into appropriate CALL statements in the host language prior to submission to the high level language compiler.

4.5 EVALUATION OF FORMATTED SYSTEMS

The principal drawback with formatted systems is that relationships are expressed explicitly and are predefined. In the relational model described in the following chapter, the basic data structure is pre-defined, but record relationships are not defined until they are used. This ability to dynamically define relationships, combined with the basic simplicity of the relational model, makes it potentially more flexible and user friendly than either the hierarchical or network approaches.

REFERENCES AND BIBLIOGRAPHY

[4.1] Codasyl A., "CODASYL Data Base Task Group Report", Proceedings of the Conference On Data Systems Languages (CODASYL), Pennysylvania, April 1971, pp 33 - 41.

[4.2] Cardenas A.F., "Data Base Management Systems", Allyn and Bacon, Inc., Second Edition, 1985.

[4.3] Date C.J., "An Introduction to Database Systems", Addison-Wesley Publishing Company, Philippines, 1981, pp 32 - 47.

[4.4] Deen S.M., "Fundamentals of Data Base Systems",The Macmillan Press Ltd, 1977, pp 23 - 37.

[4.5] Hillingdale S.H., and Tootil G.C., "Electronic Computers", Pelican, London, 1970.

[4.6] Martin J.,"Principles of Data-Base Management", Prentice-Hall, 1976.

[4.7] McFadden F.R., and Hoffer J.A., "Data Base Management", Prentice-Hall, 1978.

[4.8] Sundgren B., "Data Base Design in Theory and Practise, Towards an Integrated Methodology", Issues in Data Base Management, Stockholm Sweeden 1979, pp 121 - 135.

[4.9] Weber H., and Wasserman A.I., Editors, "Proceedings of the Fourth International Conference on Very Large Data Bases, 13-15 September, 1978, West Berlin, Germany.

CHAPTER 5

THE RELATIONAL MODEL

5.0 INTRODUCTION

A relation is a mathematical term for a two-dimensional table consisting of rows and columns, with each entry comprising a data item value. Unlike a matrix, which is homogeneous in its rows and columns, a relation is only homogeneous in its columns and not in its rows.

The idea of developing data bases based on relations started in the late 1960s, but all the systems developed then were special-purpose systems which did not have any general data-processing characteristics. The idea of a generalised relational Data Base System came up in 1970 from E.F. Codd, reference [6.4]. His idea was to develop a system which would provide data independence and data consistency both of which are difficult to achieve in the formatted Data Base Systems.

As already mentioned in Chapter 4, in formatted data bases, the data structures are designed to meet the access requirements, which implies that a change in access requirements necessitates a change in the data structures. Fortunately, in the relational model such considerations are not necessary since the access paths are universal, that is, any data item value, or any set of data item values, can be retrieved from one or more relations with equal ease. This ease of access is achieved by:

- expressing relations in what is known as the third normal form which is described in Section 5.4.2
- using a powerful data retrieval language based on relational algebra. as outlined in Section 5.3.

5.1 BASIC CONCEPTS OF A RELATIONAL DATA BASE

In relational literature, a relation, as already mentioned, is a table. The rows of the table are called tuples while each column contains attribute values belonging to the attribute domain (Figure 5.1). The number of columns defines the degree of the relation and the number of rows its cardinality.

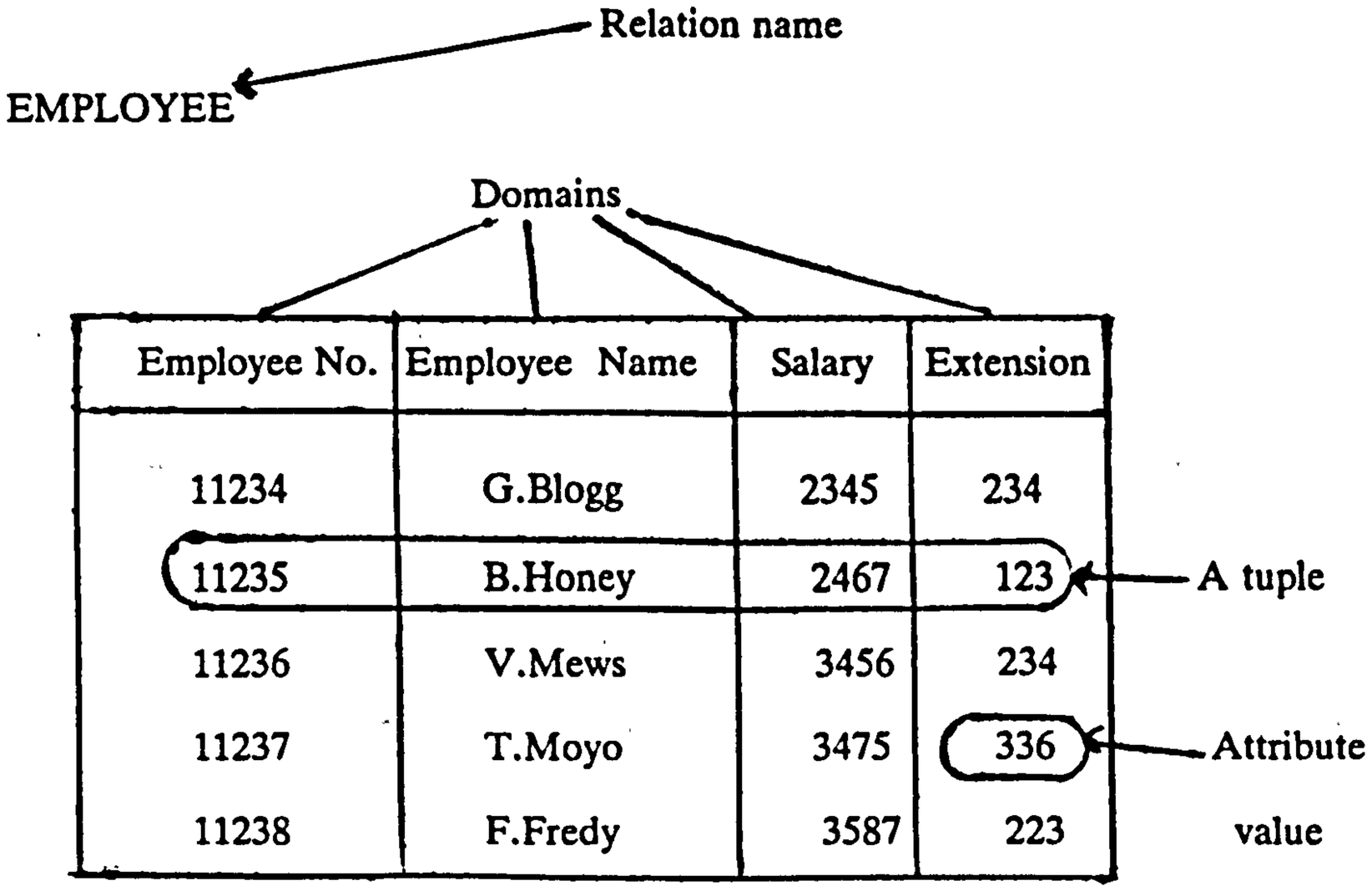


Figure 5.1 Component parts of a relation

5.2 PROPERTIES OF THE RELATIONAL MODEL

The major properties of a relation in a relational data base are:

- the intersection of each row and column contains a single attribute value, that is, multiple values are not permitted;
- the ordering of The domains is not significant. this is achieved by insisting

that each column within a relation has a distinct domain name;

- the ordering of the tuples is immaterial; that is, the rows can be interchanged without affecting the information content of the relation;
- each tuple in a relation must be unique; no two rows can have the same attribute values throughout. The significance of this property is that a row can always be uniquely identified by quoting an appropriate combination of attribute values.

5.3 DATA MANIPULATION LANGUAGES FOR THE RELATIONAL MODEL

A Data Manipulation Language (DML) is the language which the programmer uses to cause data to be transferred between his program and the data base. The DML is not a complete language by itself but relies on a host programming language to provide a framework for it and to provide the procedural capabilities required to manipulate data. Basically there are three principal approaches to the design of languages for expressing queries about relations. The notation for expressing queries is usually the most significant part of a DML. The nonquery aspects of a relational DML, or "Query Language" are often straightforward, being concerned with the insertion, deletion and modification of tuples. On the other hand, queries, which are arbitrary functions applied to relations often use a rich, high-level language for their expression.

Query languages for the relational model break down into two broad classes:

- Algebraic languages, where queries are expressed by applying specialised operations to relations, and
- Predicate calculus languages, where queries describe a desired set of tuples by specifying a predicate the tuples must satisfy.

The calculus based languages can be further divided into two classes, depending on whether the primitive objects are tuples or are elements of the domain of some attribute, making a total of three distinct kinds of query languages. Examples of these languages are:

- ISBL - Algebraic language;
- QUEL - A tuple calculus language which is the data manipulation language in INGRES and is described in Chapter 6.
- Query-By-Example - A domain calculus language.

5.3.1 Relational Calculus

The idea of using predicate calculus as the basis for a query language originated from Kuhns, reference [5.4]. The concept of a relational calculus (that is, an applied predicate calculus specially tailored to relational databases) was first proposed by Codd, reference[5.2]. Codd further presented a language explicitly based on this calculus, Data Sublanguage Alpha (DSL Alpha), reference[5.2]. Although DSL Alpha was never implemented, a language similar to it, QUEL, was developed and is the query language in the system INGRES.

A fundamental aspect of the calculus of Codd and of languages based on it, is the idea of tuple variables. A tuple variable is a variable that "ranges over" some named relation, that is, a variable whose only permitted values are tuples of that relation. For example, if the tuple variable S ranges over relation R, then, at any moment, S represents some individual tuple of R.

An alternative relational calculus, the "domain" calculus, in which instead of tuple variables, are domain variables, that is, variables that range over the underlying domain instead of relations was proposed by Lacroix and Pirotte, reference [5.5]. An example of the implementation of the domain calculus is Query By Example.

5.4 RELATIONAL DATA BASE DESIGN

One of the reasons for designing a data- base oriented system is that it is hoped to be much more viable than a conventional system. However, this does not mean that the design of a data base will not have to be changed, but it is hoped that changes will be easier to carry out when they are needed.

When designing a relational database, we are often faced with a choice among alternative sets of relation schemes, with some choices more convenient than others as demonstrated by the following example.

SUPPLIERS Relation

NAME	ADDRESS	ITEM	PRICE
Bloggs	3 Dew Rd.	nuts	450
Smith	2 Fred Pl.	bolts	230
Jones	4 Gut Ter.	wine	270
Fraser	3 Dom Drive	bricks	465

Consider the relation SUPPLIERS indicated above, with attributes NAME, ADDRESS, ITEM, PRICE. Each supplier has a name, an address, and charges a price for the item they supply.

This relation has undesirable features associated with it such as:

- redundancy - The address of the supplier is repeated once for each item supplied;
- potential inconsistency (update anomalies) - As a consequence of the redundancy, we could update the address for a supplier in one tuple, while leaving it fixed in another. Thus we would not have unique address for each supplier as we intuitively feel we should;

- insertion anomalies - We cannot record an address for a supplier if the supplier does not currently supply at least one item. We might put null values in the ITEM and PRICE components of a tuple for that supplier, but would we remember to delete the tuple with the null value? Worse still, ITEM and SNAME form a key for the relation, and it might be awkward or impossible to look up tuples with null values in the key;
- deletion anomalies - The inverse problem to insertion anomaly is that should we delete all items supplied by one supplier, we inevitably lose track of his address.

The above problems can be eradicated by splitting the relation into two relation schemes

SA(NAME,ADDRESS)

SIP(NAME,ITEM,PRICE)

The first relation scheme, SA, gives the address for each supplier exactly once; hence there is no redundancy. Furthermore, we can enter an address for a supplier even if he currently supplies no items. The second relation scheme, SIP, gives the names of the suppliers, the items they supply, and the price each supplier charges for each item.

However, there does arise some problem with this decomposition. To find the address of the supplier of an item, we must now take a join which is expensive, while with the single relation SUPPLIERS, we could simply do a selection and projection. The idea of decomposition of relations as in the above example illustrates a process known as normalisation which is described in the following section. It is worth noting that the process of normalisation is useful in the design process but is not a panecia. However, familiarisation with the theory of normalisation is essential, although the design should not be based solely on normalisation principles alone.

5.4.1 Normalisation

Normalisation is a step-by-step process of elimination of certain undesirable features from an initial unnormalised relation. A normalised relation is one which has the properties outlined in Section 5.2. The process is often described in terms of what are known as normal forms. A relation is said to be in a certain normal form if it satisfies certain constraints. For example, a relation which has the properties described in Section 5.2 is said to be in the first normal form.

A number of different properties, or "normal form" for relation schemes with dependencies have been defined. The most significant of these are called "third normal form" and "Boyce-Codd normal form". These guarantee that most of the problems of redundancy and anomalies outlined in Section 5.3 do not occur.

A full account on normalisation can be found in references [5.1,4.3,5.3]. Before describing the two normal forms, is is necessary to define a few terms.

Keys and Attributes

Within a given relation there is frequently one attribute with values that are unique within the relation and can thus be used to identify the tuples of that relation. Such an attribute is said to be the primary key of that relation. It sometimes happens that more than one attribute or a set of attributes could be the key of a record. Such alternate choices are referred to as candidate keys.

An attribute that forms a part of a candidate key is referred to as a prime attribute of the tuple and the other attributes are said to be nonprime. Consider a tuple of a relation BRIDGES with the following attributes: bridge code, region code, route, No. of arches. Each bridge has a code number, is located in a region with code number, has a certain number of arches and lies on a certain route. The bridge can be uniquely identified by either bridge code + region code, or region code + route, which are the two

candidate keys. The nonprime attribute in this case is No. of arches while the remaining attributes are prime attributes.

Functional Dependence

In attempting to lay out the relationships between data items, concern over which data items depend on which other is of significance. Functional dependence can be best described by considering two data items, A and B, which belong to a record R. B is said to be functionally dependent on A if A identifies B, that is, knowing the values of A, the values of B associated with A can be found. Consider the student relation

STUDENT(NAME,ADDRESS,AGE)

with attributes in parenthesis. NAME apparently determines ADDRESS in the relation and there is said to be a "functional dependence" of ADDRESS on NAME. Furthermore, a data item or a collection of data items, B is said to be fully functionally dependent on another collection of data items, A, if B is functionally dependent on the whole of A but not on any subset of A.

Transitive dependence

A record may have a data item which is not a key but which itself identifies other data items. This is referred to as a transitive dependence.

Suppose A, B and C are three data items or distinct collection of data items of record R. If C is functionally dependent on A and B is functionally dependent on A, it follows that C is functionally dependent on A. However, if A is not functionally dependent on B or B is not functionally dependent on C, then C is said to be transitively dependent on A.

Consider the following two examples relating some part numbers named A, B, C, and so on.

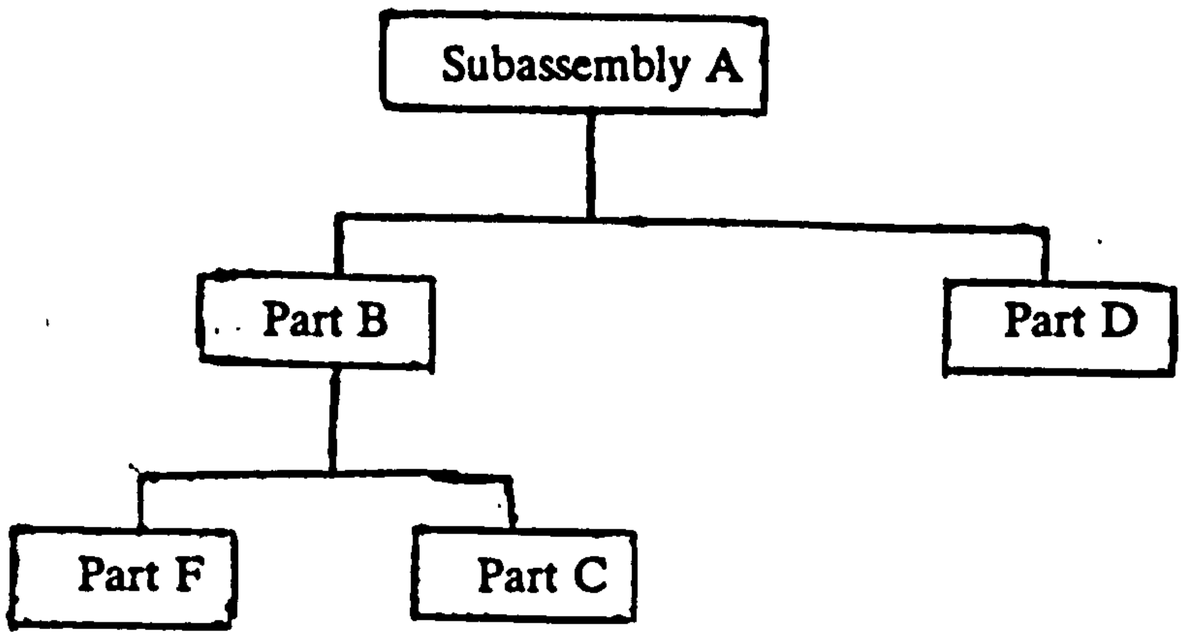


Figure 5.2 A and B are transitively (directly) dependent on B and C respectiely, but A is intransitively (indirectly) dependent on C.

In Figure 5.2, part B is a direct component of subassembly A and part C is a direct component of part B, but part C is not a direct component of subassembly A. Subassembly A is therefore directly or intransitively dependent on part C, part B being intransitively dependent on part C. However, in Figure 5.3 part C is made a direct component of both subassembly A and part B, and therefore subassembly A is now intransitively dependent on both part B and part C, the relationship between part B and C remaining unchanged.

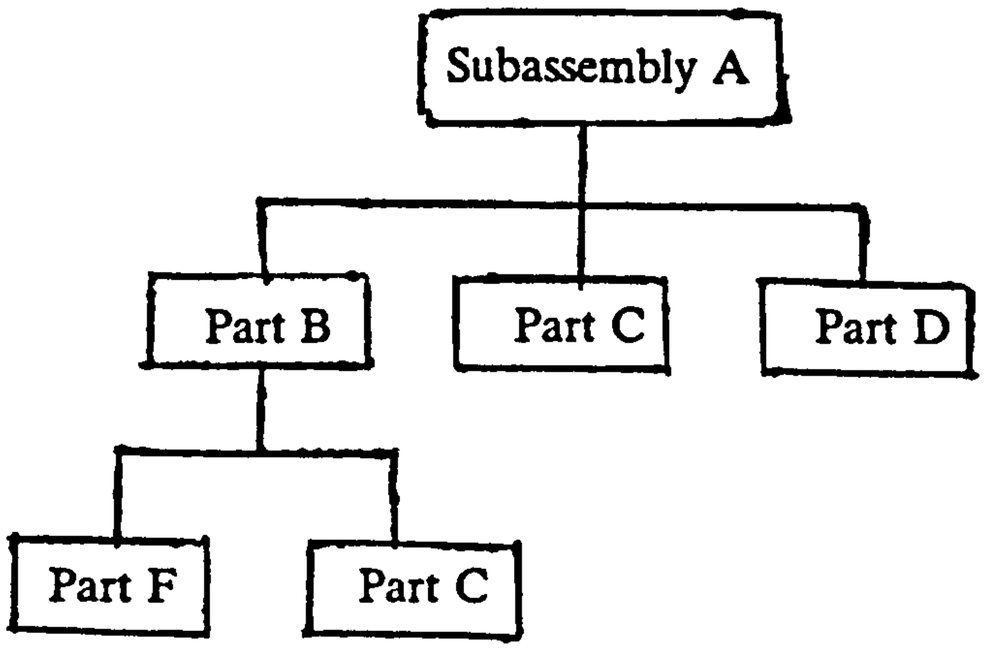


Figure 5.3 Both A and B are transitively dependent on C

5.4.2 Third Normal Form (3NF)

A normalised relation is said to be in third normal form if all its nonprime attributes are nontransitively and fully dependent on each primary key. Consider the relation schema

HOUSES(BUILDER,STYLE,PRICE)

that is, a house is built by a builder who charges a certain price for a style of the house. This relation is not in 3NF, since the nonprime attribute PRICE is transitively dependent on the primary key, BUILDER. The transitive dependency can be eliminated by splitting the relation into two relations,

HOUSES(BUILDER,STYLE)

COST(STYLE,PRICE)

These two relations cannot contain any transitive dependencies since they are each only of degree two.

5.4.3 Boyce/Codd Normal Form (BCNF)

A normalised relation is in BCNF if and only if every determinant is a candidate key. It is worth noting that the definition of BCNF refers to candidate key, not just the primary key as is the case with 3NF. The 3NF definition does not satisfactorily handle the case of a relation with two or more composite and overlapping candidate keys. From the definition of BCNF, the following can be asserted:

- all nonprime attributes must be fully dependent on each key;
- all prime attributes must be fully dependent on all keys of which they are not part;
- no attribute (prime or not) can be fully dependent on any set of attributes that is not a key.

Since each key is fully dependent on every other key, the keys are functionally equivalent. That is, for every pair of keys there are functions in both directions connecting them. By the first and last restrictions, all nonprime attributes are dependent only on the keys. Consider a case in which a value for a nonprime attribute is altered in some tuple. Since the only functional dependencies that exist for that attribute are those mapping a key into that attribute and since each key value is unique within the relation, there can be no other tuples affected by the update. Thus, the value of each nonprime attribute in a tuple is independent with respect to all other tuples.

Prime attributes, have no identity of their own within functional dependencies, but rather exist only as part of some key. Altering the value of a prime attribute in a tuple is therefore equivalent to altering the value of a key. Consider the case in which a value for a prime attribute is altered in some tuple. Equivalently, it can be assumed that a key value for that tuple has been altered. However, by definition, all keys are distinct, hence this change can only affect the tuple whose key value was changed. All other tuples have key values different from the one that was altered and therefore cannot be affected.

This point can best be illustrated by an example in which such an alteration can matter. In the example the notation A→B, that is, A functionally determines B, is used as abbreviation. Consider a relation R(A,B,C.D) with AB→C and B→D as the functional dependencies in the relation. AB is the only key for R. If a value of B is changed in one tuple, this changes the key for that tuple. However, it is possible that some other tuple is affected, since some other key value may have the same value of B as the one that was altered. The discrepancy can be eliminated by replacing B→D by AB→D. Changing a value of B is now equivalent to changing the key value. Since by definition no other tuple can have the same key value, no other tuple can be affected.

5.5 EVALUATION OF THE MODELS

The evaluation of the three models requires that a criteria by which they should be judged be stated initially. The two primary concerns are:

- Ease of use - It is essential for the model that makes accurate programming and the phrasing of queries easy.
- Efficiency of implementation - For large databases, the cost of storage space and computer time dominate the overall cost of implementing a database. The need is for a data model in which it is easy for the DBMS to translate a specification of the conceptual-to-physical mapping into an implementation that is space efficient and in which queries can be answered efficiently.

The relational model is in no doubt the superior by the criterion of ease of use. It provides only one construct that the programmer or user must grasp, that is, the relation. Furthermore, as already discussed in Section 5.3, there are rich, high level languages for expressing queries on data represented by the relational model. These languages make systems based on the relational model available to persons whose programming skill is not great.

In comparison, the network and hierarchical models require the understanding of both record types and links and their interelationships.

Considering the potential for efficient implementation, the network and hierarchical models score high. This is due to the fact that many-to-many mappings are not efficiently implemented. Relations can and often do, represent many-to-many mappings. However, some specialised data structures can be used to implement relations, as well.

The level of the Data Manipulation Language (DML) can profoundly affect the ease with which a DBMS can be used, just as it is easier to program in Fortran than in

Assembly language. Relational DBMS's have stressed languages of very high level, while DBMS's based on the other models have tended to have languages of lower level. One of the high-level relational languages, QUEL, is described in Chapter 6.

In the past, commercial database systems have undoubtedly been almost uniformly based on the formatted data base models because the emphases of such systems has been on the maintenance of large data bases, and these models lend themselves most easily to the necessary efficient implementation. However, with the latest attention the relational model has obtained recently, it has become clearer that the same concepts used to design large data bases apply as well to small and medium scale data bases, and there are many more small databases than large ones.

Second, many of the apparent inefficiences of the relational model can be eliminated. Some of the optimisation techniques for relational data manipulation languages that allow these languages to use time efficiently have been developed. A full account of these techniques can be found in reference [5.8]. Research aimed at producing good physical implementations of relations is also currently underway.

Two well known relational data base systems that have been developed are system R (IBM,San Jose) and INGRES (University of California,Berkley) which was used in the work described in this thesis. Chapter 6 describes the INGRES system.

5.6 EXAMPLES ON MASONRY BRIDGES

An example of attribute keys that fully define a bridge are its location and the route on which the bridge lies. On the other hand the name of the bridge is not a key attribute as different bridges could have the same name. Also, the location on its own is not necessarily a key attribute as there could be two or more bridges on the same location.

The location of a bridge defines its geology and hence the material of construction it is likely to be constructed on. Also, the route on which the bridge lies is related to the

traffic the bridge will carry. Hence, in choosing the parameters that best describe a bridge before any information can be relied upon as describing that particular bridge, enough qualifications should be added to the retrieval statement (see Chapter 9).

The defects that may be found on a bridge are dependent on the conditions of the various components of the bridge. For example, the cracks on the arch barrel may be due to the movements of the spandrel walls (see Section 2.3.3). Also, the maintenance details are dependent on the authority on which the bridge is located.

In Figures 5.2 and 5.3, the Subassembly can be thought of as the various components that make up the bridge. The dependences of the various defects on the component parts of the bridge are as outlined in Section 2.3.

REFERENCES AND BIBLIOGRAPHY

[5.1] Codd E.F., "Further normalisation of the data base relational model", in Data Base Systems, edited by R.Rustin, Prentice Hall, Englewood Cliffs, New Jersey.pp 33-64

[5.2] Codd E.F., "Relational Completeness of Data Base Sublanguages", In Data Base Systems, Courant Computer Science Symposium 6 Series, Vol. 6, Englewood Cliffs, NJ., Prentice Hall, 1972, pp 73 - 79, pp 67 - 70.

[5.3] Howe D.R., "Data Analysis for Data Base Design", Edward Arnold (Publishers) Ltd, 1983. Sons, New York, 1979, pp 111 - 123.

[5.4] Kuhns J.L., "Answering Questions By Computer; A Logical study," Report RM -5428 -PR, Rand Corp., Santa Monica, California, 1967, pp 18 -25.

[5.5] Lacroix M., and Woton P., "A Comprehensive Formal Query Language for a Relational Data Base", R.A.I.R.O. Informatique/Computer Science 11, No.2, 1977, pp 11 - 17.

[5.6] Tsichritzis D.C., and Lochovsky F.H.,"Data Base Management Systems", Academic Press, Inc., New York, 1977.

[5.7] Tsichritzis D.C., and Lochovsky F.H., "Data Models", Prentice Hall, 1982.

[5.8] Ullman J.D.,"Principles of DATABASE SYSTEMS",Computer Science Press,Inc.,1982, pp 97 - 113.

CHAPTER 6

THE INGRES RELATIONAL DATA BASE MANAGEMENT SYSTEM

6.0 BACKGROUND

INGRES (Interactive Graphics and Retrieval System) is a relational database system which is implemented on top of the UNIX† operating system developed at Bell Telephone Laboratories, reference [6.14]. Figure 6.1 shows INGRES's relationship with its UNIX neighbourhood. The UNIX process which runs as the front end is described in later sections of this Chapter.

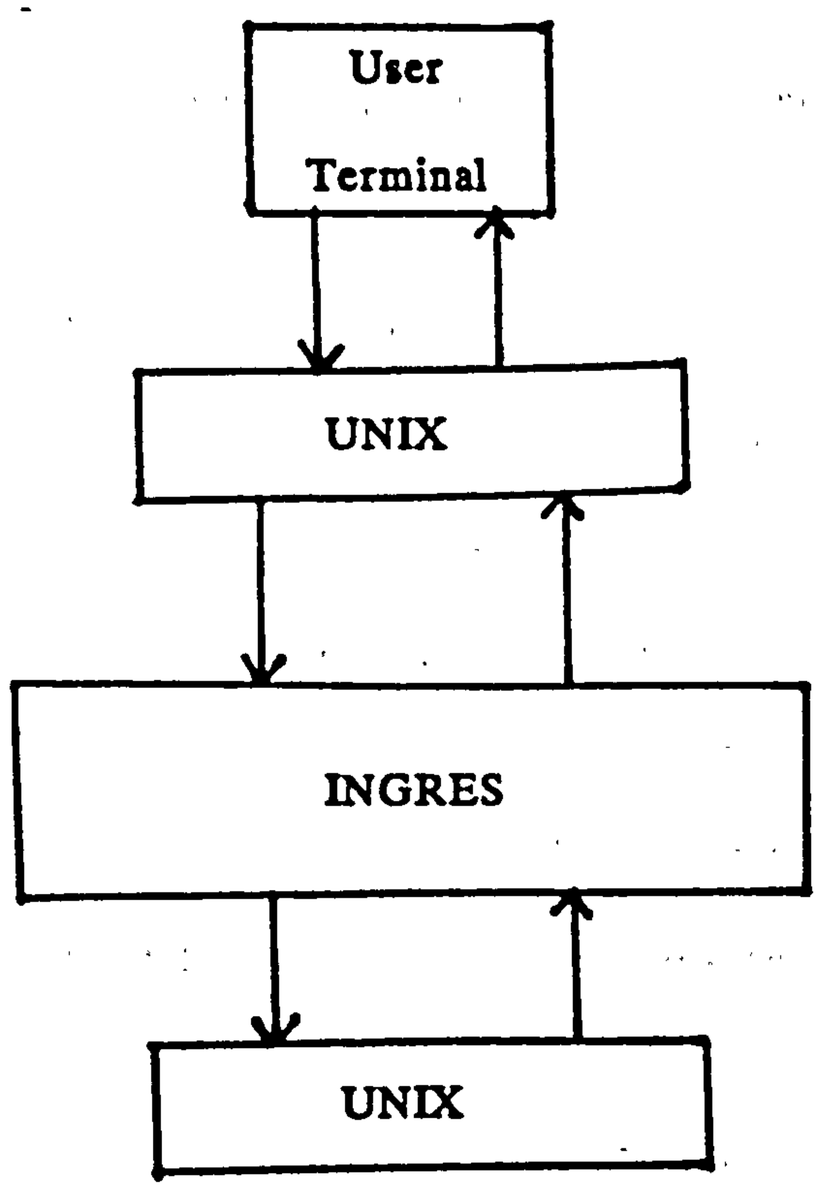


Figure 6.1 The INGRES Environment

† UNIX is a trademark of Bell Telephone Laboratories, Inc.

The implementation of INGRES is primarily programmed in C, a high level language in which UNIX itself is written. Complete descriptions of the INGRES data management system are contained in references [6.9,6.11,6.13].

The advantages of a relational model for database management systems are extensively discussed in references [6.4,6.6,6.7]. In choosing the relational model, the particular motivation was the high degree of data independence that such a model offers, and the provision of an entirely procedure-free facilities for data definition, retrieval, update, access control, support of views and integrity verification.

6.1 USING THE INGRES RELATIONAL DATA BASE MANAGEMENT SYSTEM

INGRES is designed to support an Information System and to enable end-users to access and control that system, either

- directly by using a terminal as in Figure 6.2(a) or
- indirectly via an application program as in Figure 6.2(b).

From the point of view of using the system, it is well to recognise that these two methods of using the system are more or else equivalent. In the first case an end-user interacts with the system directly, while in the second case a programmer (who is a system designer) has to interact with the system. In both cases the facilities offered to each user group are more or less the same, and for this reason the system supports a data sub-language (QUEL) and an embedded data sublanguage (EQUEL). This thesis is mainly about the area in the dotted box in Figure 6.2(b).

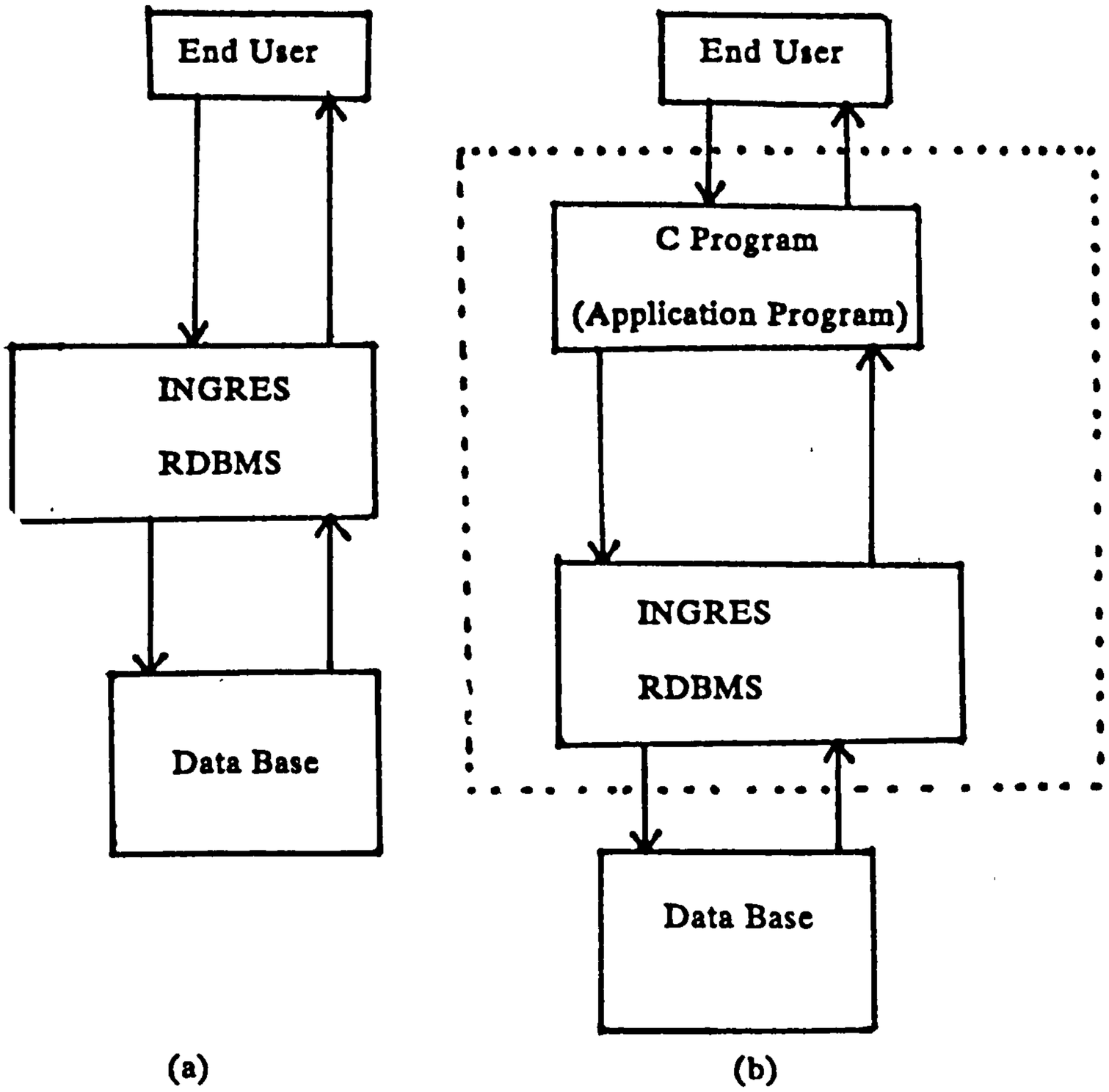


Figure 6.2 Using the INGRES Relational Data Base Management System

6.2 QUEL AND INGRES UTILITY COMMANDS

QUEL is a complete INGRES query language which frees the programmer from concern for how data structures are implemented and what algorithms are operating on stored data reference [6.5].

A QUEL interaction includes at least one RANGE statement of the form:

RANGE OF variable-list IS relation-name.

The purpose of this statement is to specify the relation over which each variable ranges. The variable-list portion of a RANGE statement declares variables which will be used

as arguments for tuples (tuple variables).

Furthermore, an interaction includes one or more statements of the form:

Command [result-name] (target-list)

[where Qualification]

where "Command" is either RETRIEVE, APPEND, REPLACE, or DELETE. For RETRIEVE and APPEND, the "result-name" is the name of the relation which qualifying tuples will be retrieved into or appended to. For REPLACE and DELETE, "result-name" is the name of a tuple variable which, through the qualification, identifies tuples to be modified or deleted. The target-list is a list of the form:

result-domain = QUEL Function....

The result-domains are domain names in the result relation which are to be assigned the value of the corresponding function. Appendix 6.A shows some examples which demonstrate some valid QUEL interactions for each of the QUEL commands. A complete description of the QUEL language can be found in reference [6.11].

In addition to the above QUEL commands, INGRES supports a variety of utility commands. These can be classified into the following major categories:

- invocation of INGRES;
- creation and destruction of databases;
- creation and destruction of relations;
- copy of data to and from INGRES;
- storage structure modification;
- miscellaneous.

A description of each of these utility commands is given in APPENDIX 6B.

6.3 EQUEL

Although QUEL alone provides the flexibility for many data management requirements, there are some applications which require the flexibility of a general purpose programming language in addition to the database facilities offered by QUEL. To this end, has been implemented a new language, EQUEL (Embedded QUEL) which consists of QUEL embedded in the general purpose programming language C as illustrated in Figure 6.3.

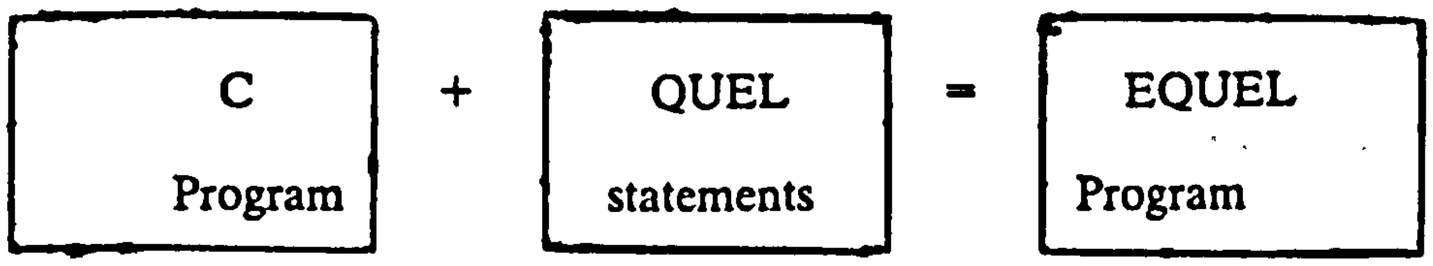


Figure 6.3 From Quel to EQUEL

6.4 The INGRES PROCESS STRUCTURE

As already mentioned in Section 6.1, INGRES can be invoked in two ways. One way is by direct invocation from UNIX, and the other is by calling it (INGRES) from a host language. These two methods of interaction with INGRES are discussed in turn, but before doing so, a few details will be said about INGRES.

6.4.1 UNIX

Processes in UNIX may create child processes by using the UNIX "fork" system call which creates an exact duplicate of the parent process. There is only one difference between the two processes, the child process has has a zero number process identity (pid), while the parent process always has a positive number as its process identity. These processes may communicate with each other via an interprocess communication facility called "pipes". A pipe is a one direction communication link which is written

into one process and read by the second one. Synchronisation of pipes is maintained by UNIX, hence no messages are lost.

Furthermore, each process has a standard input device and a standard output device. These are usually the user's terminal but may be directed by the user to files, pipes to other processes or other devices. A full account on the UNIX system can be found in reference [6.20].

6.4.2 Invoking INGRES from UNIX

Issuing INGRES as a UNIX command causes the process structure shown in Figure 6.4 to created.

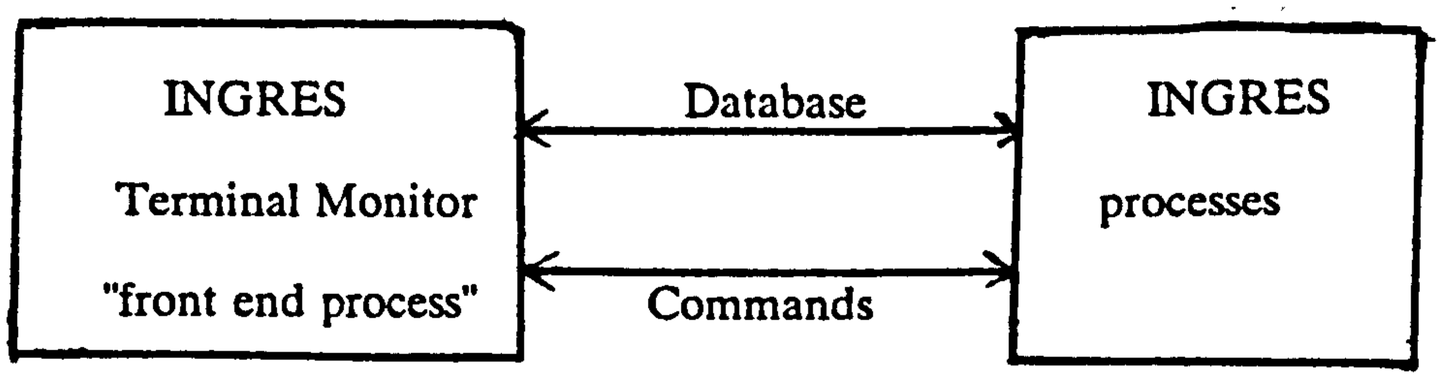


Figure 6.4 INGRES Structure (Direct Interaction)

The INGRES interactive terminal monitor allows the user to formulate, print, edit, and execute collections of INGRES commands. The monitor maintains a workspace with which the user interacts until satisfied with the interaction. The contents of the workspace are then passed down to the INGRES process when execution is desired. The full set of commands accepted by the current INGRES terminal monitor is given in reference [6.13].

6.4.3 Indirect Interaction with INGRES

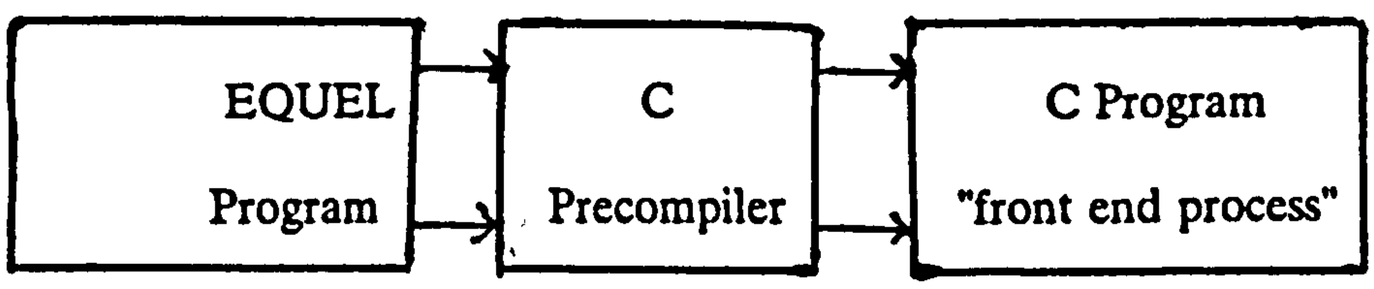
In an indirect interaction with INGRES there is a need to call INGRES from a host language. This can be achieved in three separate ways which are:

(i) altering the compiler to accept INGRES commands

(ii) writing a precompiler to convert QUEL statements and

(iii) writing a subroutine call interface.

Altering the compiler is clearly a major task. In order to implement EQUEL, a precompiler was developed to convert an EQUEL program into valid C program with QUEL statements converted to appropriate C code and calls to INGRES.



The resulting C program is then compiled by the normal C compiler, producing an executable module. It is worth noting that when an EQUEL program is run, it is this executable module that is used as the front end process instead of the interactive terminal monitor as shown in Figure 6.5.

A full account on the functions of the C precompiler can be found in reference [6.1].

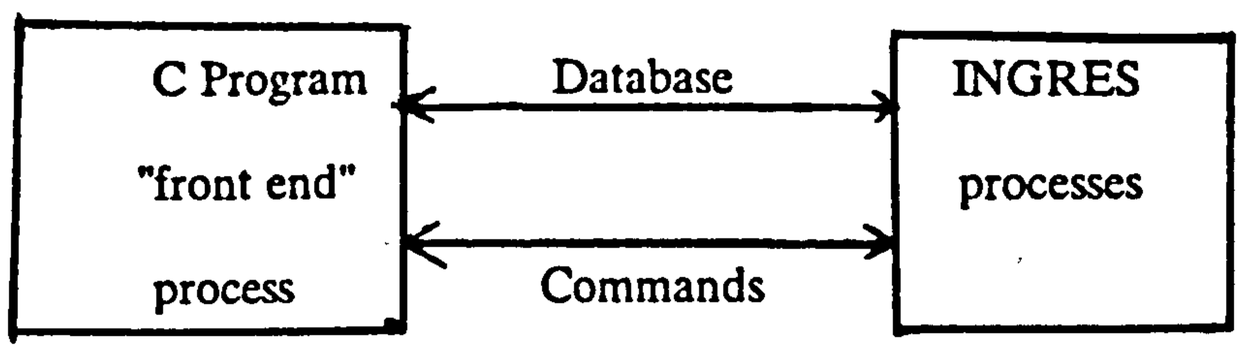


Figure 6.5 The INGRES Structure (Indirect Interaction)

A condition code is returned through the UNIX interprocess message systems to indicate success or the type of error encountered. The functions performed by the EQUEL translator are fully discussed in reference [6.1].

The subroutine call interface works in a similar manner as an EQUEL program. During execution of the front-end program, database commands (QUEL statements in the C program) are passed between the application program and the INGRES process over the UNIX interprocess message systems (pipes) and are processed by INGRES.

6.5 INGRES STORAGE STRUCTURES

INGRES supports three storage structures for a relation, namely, heap, hash and Isam. A brief description of each of these storage structures will be given in this section. A detailed description can be found in reference [6.9]. Any user created relation can be converted to any of the storage structures outlined below by using the "modify" command (Appendix 6B).

6.5.1 Heap

When a relation is first created, it is created as a heap, that is, an unordered collection of tuples. A relation stored as heap has duplicate tuples and the location of the tuples is unknown. Hence, query processing involves the scanning of the entire relation in a logically sequential order, one page at a time. This type of storage structure is only suitable for:

- very small relations, where the added cost of other storage structures is unjustifiable;
- transitional storage of data as is the case when data is being moved into or out of the system by COPY;

- certain temporary relations created as intermediate results during processing a query.

6.5.2 Hash

Hashing is a direct-accessing technique in which the key to a domain is converted to a pseudo-random number from which an address is derived for the required tuple. A hashed relation contains no duplicate tuples.

This mode of storage is well suited for situations whereby access is to be conditioned on an exact value of a qualification. Using the hashing the main page of the tuple satisfying the qualification is identified. That particular page is then scanned through to identify the required tuple.

In the case where more tuples hash to a location than can fit on one page, overflow pages are created and are linked to the original page using pointers.

6.5.3 Isam (Indexed Sequential Accessing Method)

In a relation which is Isam, the index associated with the relation is first searched to determine the page that the the required tuples would fall on, and processing follows as in a hashed relation. Duplicate tuples are also removed in this type of storage.

This mode of storage is suitable for locating tuples referenced in a qualification by both exact values and range of values. An Isam structure is never as efficient as a hash structure, since the Isam directory must be searched to locate tuples.

6.5.4 Secondary Indices

If a relation is not hash or Isam on a domain, but has a secondary index on that domain, the secondary index is searched to find the logical page number and offset

within the page of the qualifying tuples. All pages containing a qualifying tuple are read, and the data in qualifying tuples located. It is worth noting in this case that the secondary index identifies the qualifying tuples. Hence, unlike in the case of an Isam or hash structure, a data page does not have to be exhaustively searched.

6.5.5 Compression

In the above-mentioned storage structures, fixed length tuples are stored. In addition, the above-mentioned storage structures can be used in conjunction with data compression techniques, reference [6.10], in situations where increased storage utilisation outweighs the overhead of encoding and decoding of data during access. These modes are known as compressed hash and compressed Isam. Note that it does not make sense to compress a heap structure.

REFERENCES AND BIBLIOGRAPHY

[6.1] Allman E., Held G., and Stonebraker M., "Embedding a Data Manipulation Language in a General Purpose Programming Language", Proceedings of the 1976 ACM - SIGPLAN - SIGMOD Conference on Data Abstraction, Definition, and Structure, Salt Lake City, UT, March 1976, pp. 25 - 35.

[6.2] Bratsbergsengen K., and Risnes O., "ASTRAL - a Structural Relational Applications Language", Proceedings of the SIMULA Users Conferenece", September 1977.

[6.3] Chamberlin D., et al., " A Unified Approach to Data Definition, Manipulation, and Control", IBM Journal of Research and Development, Vol. 20, No. 6., November 1976, pp. 560-575.

[6.4] Codd E.F., "A Relational Model of Data for Large Shared Data Banks", Communications of the ACM Vol. 13, No. 6, June 1970, pp. 377 - 387.

[6.5] Codd E.F., "A Data Base Sublanguage Founded on The Relational Calculus", Proceedings of the 1971 ACM - SIGFIDET Workshop on Data Description Access and Control, San Diego, CA, November 1971, pp. 35-68.

[6.6] Codd E.F., and Date C.J., "Interactive Support for Non-programmers, the Relational and Network Approaches", Proceedings of the 1974 ACM - SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, MI, May 1974, pp 33 - 42.

[6.7] Date C.J., and Codd E.F., "The Relational and Network Approaches: Comparison of the Application Programming Interfaces", Proceedings of the 1974 ACM - SIGFIDET WOrkshop on Data Description, Access and Control, Ann Arbor; MI, May 1974, pp. 84 - 113.

[6.8] Date C.J., "An Architecture for High-Level Language Database Extension", Proceedings of the 1976 ACM-SIGPLAN-SIGMOD Conference on Data Abstraction, Definition, and Structure, Salt Lake City, UT, March 1976.

[6.9] Epstein R., "Creating and Maintaining a Database Using INGRES", Memo No. M 77-71, Electronics Research Laboratories, University of California, Berkely, CA, December 1977, pp 99 - 105.

[6.10] Gottlieb D., et al., "A Classification of Compression Methods and Their Usefulness in a Large Data Processing Centre," Proceedings of the 1975 AFIPS National Computer Conference, Vol. 44, Anaheim, CA, May 1975, pp. 453-458.

[6.11] Held G., Stonebraker M., and Wong E., "INGRES: A Relational Data Base Management System", Proceedings of the 1975 AFIPS National Computer Conference, Vol. 44, Anaheim, CA, May 1975, pp. 409-416

[6.12] Hutt A.T.F., "A Relational Data Base Management System," John Willey & Sons, New York, 1979.

[6.13] INGRES Version 6.2 Referenece Manual, Memo No. M79-43, Electronics Research Laboratories, University of California, Berkely, July 1979, pp 112 - 123.

[6.14] Ritchie D., "C Reference Manual, Bell Telephone Laboratories, Murray Hill, NJ,1974, pp 23 - 27, pp 45 - 51.

[6.15] Rowe L., and Shoens K., "Data Abstraction, Views and Updates in RIGEL", Proceedings of the 1979 ACM-SIGMOD Conference on the Management of Data, Boston, MA, June 1979.

[6.16] Schmidt J., "Some High Level Language Constructs for Dat of Type Relation," ACM Transactions on Database Systems, Vol. 2, No. 3, September 1977, pp. 247-261.

[6.17] Stonebraker M., Wong E., Kreps P., and Held G., "The Design and Implementation of INGRES," ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976, pp. 198 - 222.

[6.18] Stonebraker M., "A Functional View of Data Independence", Proceedings of the 1974 ACM - SIGFIDET Workshop on Data Description and Control, Ann Arbor, MI, May 1974.

[6.19] Wong E., and Youssefi K., "Decomposition: A Strategy for Query Processing" ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976, pp. 223 - 241.

[6.20] UNIX Version 4.2 Reference Manual, Electronics Research laboratory, University of California, Berkely. Electronics Research Laboratory, University of California, Berkely, April 1975, pp 36 - 45, pp 63 - 71.

[6.21] Stonebraker M., Editor, "The INGRES Papers, Anatomy of a Relational Database System", Addison-Wesley Publishing Company, 1986.

CHAPTER 7

DEVELOPMENT OF THE INTERFACE

7.0 BACKGROUND

As already mentioned in Chapter 6 (Section 6.4.3), there are three ways of developing user interfaces. Designing a language like EQUEL as outlined in Chapter 6 faces the difficulty of attaching a database system clearly to a programming language. This results from the fact that the two environments do not have the same underlying type system. For example, the database has a type, "relation" that is not supported in the programming language environment. Moreover, one would like to pass references to tuples as arguments to programming language procedures. Such a feature is impossible as the programming language does not know the structure of a tuple returned at run time from the database. This problem can be solved by building a new programming language, which could do both general purpose computation and database access in one environment. Alternatively, one could extend a modern programming language (e.g. Pascal) with desired features. However, this is a subject for the programming language research community. Writing a preprocessor is definitely a big task. It is for these reasons and because this project is application oriented that a subroutine call interface was adopted.

The problem that had to be overcome was to attach a database system to a programming language, C. The reason for the use of C language is that this is the language upon which UNIX is developed which is the future language for computer systems. Furthermore, INGRES, is itself return in C.

7.1 BASIC STRUCTURE OF USER INTERFACE

Figure 7.1 shows the basic structure of the interface between INGRES and the user. There are three processes that make up the entire interface, the parent process, and two child processes that are spawned by the fork system call (see reference [6.20]). Communication between the three processes is established by the use of bidirectional pipes as discussed in Section 7.4.

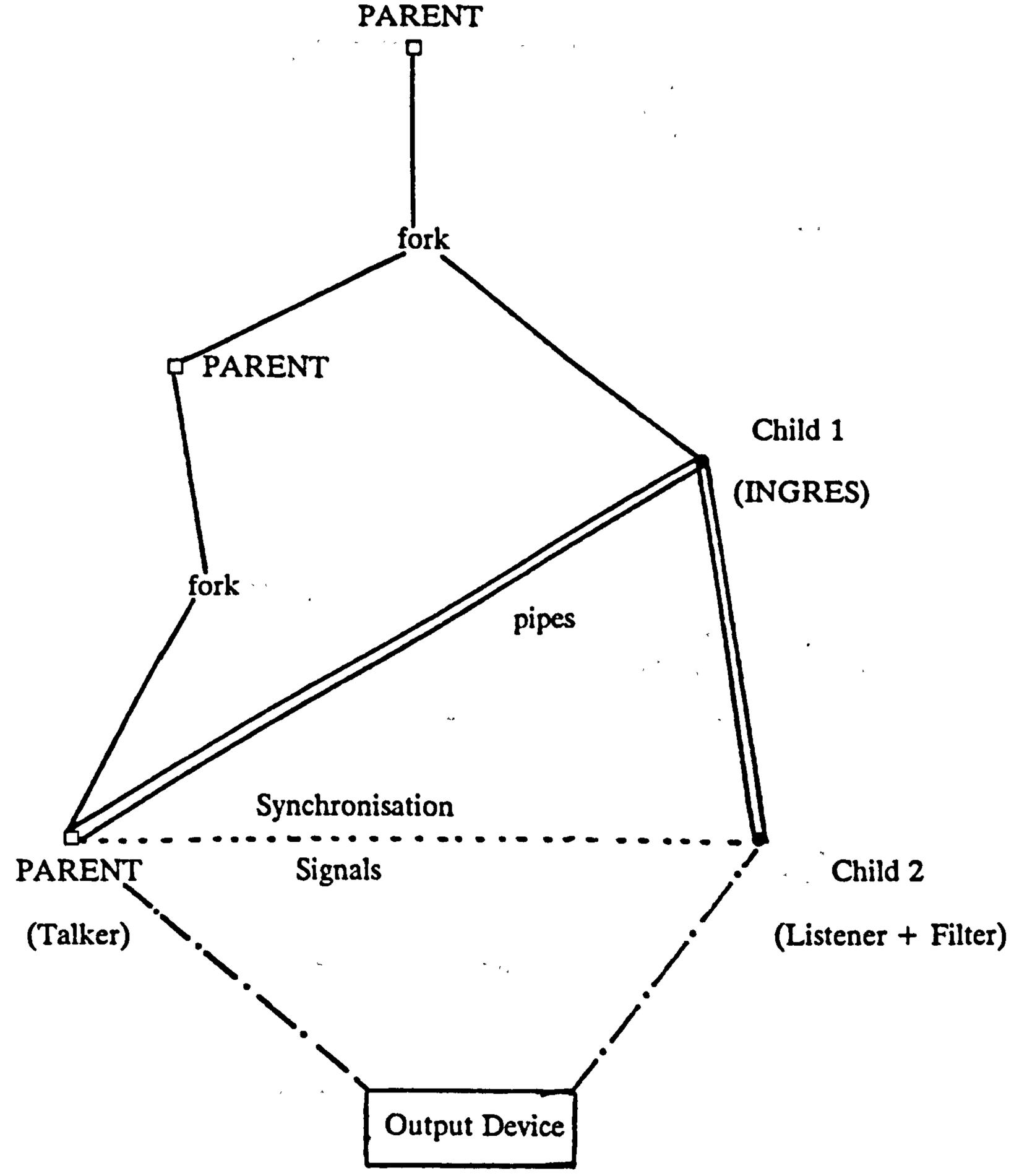


Figure 7.1 Structure of Interface.

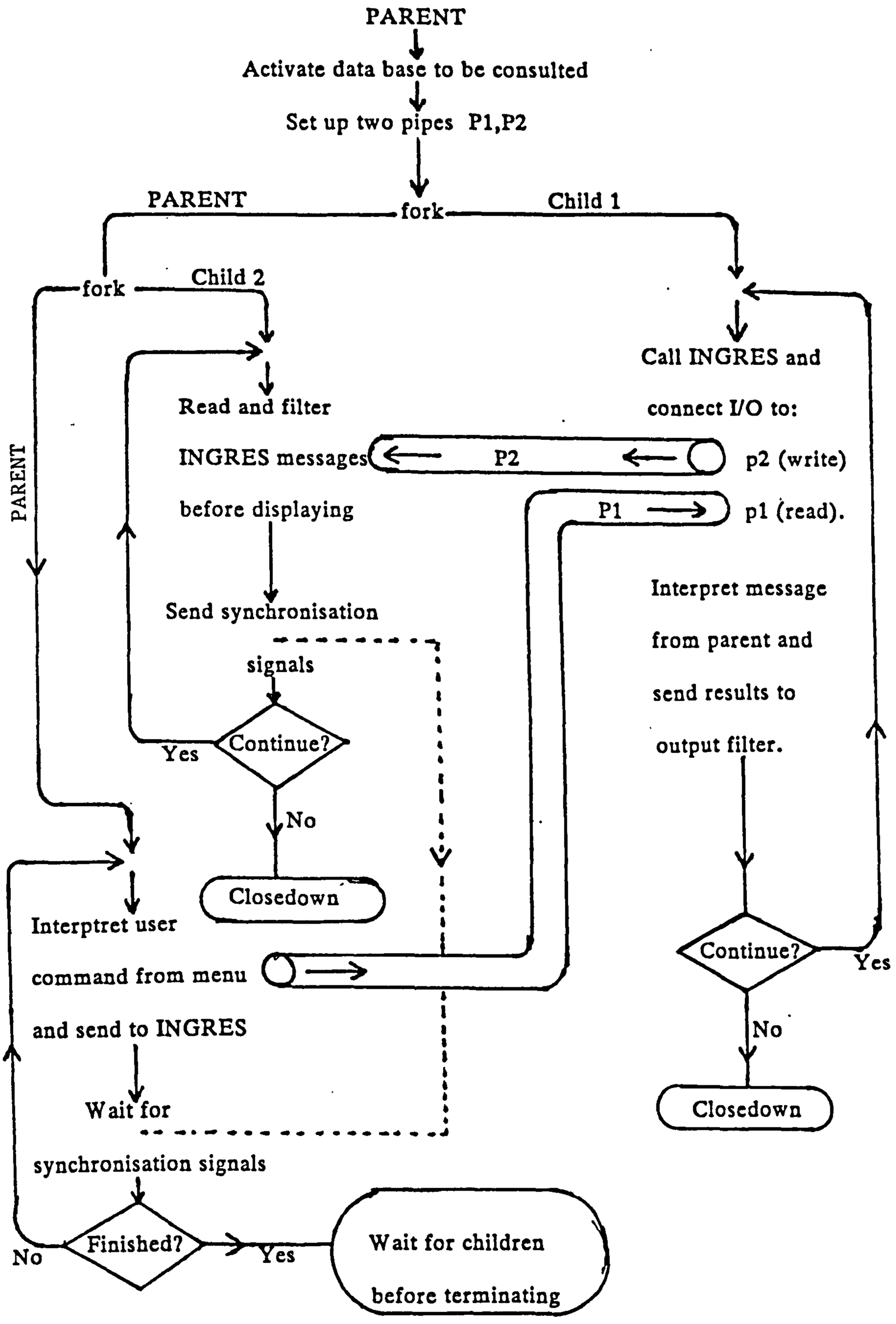


Figure 7.2 Operation of Interface.

The first call to 'fork' creates Child 1 which in turn invokes INGRES as described in Appendix 6B. The second call creates the Child 2. The roles of each of the three processes is described in Section 7.2. Synchronisation of the processes is achieved by the use of signals as outlined in Section 7.3.

7.2 ROLE OF THE THREE PROCESSES

Figure 7.2 indicates the role played by each individual process.

7.2.1 Parent Process

Besides giving birth to the two children, the parent process has the role of interpreting the QUEL and the INGRES utility commands and then sending them to Child 1, INGRES. The parent also has the added duty of establishing when the conversation among the "family" is completed and terminating the child processes when this happens.

7.2.2 INGRES (Child 1) and Child 2

Child 1 has the responsibility of accepting the instruction sets from the parent process, processing the queries asked for and sending the results to Child 2. Child 2 in turn, accepts the output from Child 1, filters the output and consequently sends them to the appropriate output device. Also Child 2 is responsible for sending synchronisation and another signals to the parent process.

7.3 PROCESS CONTROL

Signals are fundamental to process control in Unix and are the only means by which a process can control another. A signal can be considered as an interrupt which is sent to a process to cause it to either stop or to initiate some appropriate action. The latter can

be achieved by the use of subroutines containing the relevant actions to be taken as demonstrated by the example below.

```
main()        /* Main program */
{
   int flag = 0; /* Initially set control flag to zero value */
   int wait_for_ingres();   /* Tell compiler 'wait_for_ingres',
   int wait_for_signal();      'wait_for_signal' and 'func' are
   int func();                 routines */
   .
   pid = getpid(); /* Get process identity of parent process */
   pid1 = fork();
   if (pid2 = fork() == 0) wait_for_ingres();
    /* Create two child processes, Child 1 and Child 2,
       get their identities. Child 2 then listens to INGRES */
   .
   wait_for_signal(); /* Call 'wait_for_signal' routine */
   .
}
wait_for_ingres() /* Child 2 listens to INGRES */
{
       kill(pid,SIGINT); /* Child 2 sends synchronisation signals
                            to parent process */
}
wait_for_signal() /* Parent process waits for signals */
do
{
       signal(SIGINT,func);
```

```
        /* Trap synchronisation signal and cause parent process to execute
           the routine 'func' when the signal, SIGINT, is received */
}
while flag == 0;
flag = 0;        /* Reset control flag to zero value */
func()
{
        flag = 1; /* Set value of control flag to one */
}
```

A full account on signals, how they are sent and received by the processes can be found in references [6.20].

7.4 INTERPROCESS COMMUNICATION

As already mentioned in Chapter 6 (Section 6.4.1), the standard shell, sh, offers only one-way pipelines and does not offer any notation to set up two-way communication between processes, although there are experimental shells that do so. Bidirectional pipes are set up from C programs using the pipe system call as outlined below.

Interprocess channels are created using the pipe system call which creates an input/output (I/O) mechanism called a pipe. Each pipe has both the reading end and the writing end. Having created the pipes, two or more cooperating processes created by subsequent fork calls can pass data through the pipes with read and write (see reference 6.20) system calls. It is worth noting that the child processes created by subsequent fork calls inherit the communication system that is already established. The interprocess communication for this interface was established in routine 'myopen' (see Appendix 10).

7.5 OPERATION OF INTERFACE

Figure 7.2 is a flow diagram showing the operation of the interface. The initial step is to invoke the INGRES data base from within the application program. Two pipes are then created as described in Section 7.3 and the I/O mechanism of the two pipes is set up such that the reading end of pipe 2 (P2) and the writing end of pipe 1 (P1) are connected to INGRES. This leaves the writing end of P1 and the reading end of P2 open to other processes (parent and Child 2) for communication to be established between the processes.

7.5.1 Communication Between Parent and INGRES

The interface is menu driven, hence a set of QUEL commands and INGRES utility commands from a particular selection from the menu are sent to the INGRES workspace for each selection. These sets of instructions are sent in small "chunks" until the entire instruction is built up, at which moment the instruction to process the query is sent (see Appendix 6B). The example below illustrates how a query to print relations in a data base (see Appendix 6B) would be carried out by the interface.

Example 1: Print relations in a data base.

```
main()
{
        int to_ingres; /* File descriptor for writing end of P1 */
        int run_ingres();       /* Tell compiler 'run_ingres and
        int show_rels_in_db();     'show_rels_in_db' are routines */
        char param[40]; /* 'param' is type character to hold
                           a maximum of 40 characters */
        char com;  /* 'com' is of type character */
        switch(com) /* Multiple-choice control structure */
```

```
{
        .
        .
        case 'P':
        case 'p':       show_rels_in_db();
                        break;
        .
        .
        }
}
show_rels_in_dbs()
{
        sprintf(param, "help\n");
        write(to_ingres,param, strlen(param));
        run_ingres();
}
run_ingres()
{
        write(to_ingres,"\\g\n",3);
        /* Process the query, transmit to INGRES, and run */
}
```

The above example illustrates how in general, instructions from the menu (parent process) are sent to the INGRES monitor, processed, transmitted and run.

All input and output is done by two system calls, read and write, which are accessed from C by functions of the same name. For both, the first argument is a file descriptor. The second argument is an array of bytes that serves as the data source or destination. The third argument is the number of bytes to be transferred.

The function sprintf has the general format sprintf(s, format). This function places 'output' in the string s.

7.5.2 Communication Between INGRES and Child 2

Having processed the query, INGRES writes the results onto the write end of P2. Child 2 in turn reads from the read end of P2 the results from INGRES and filters the results and finally sends them to the output device. The example below illustrates how this is achieved.

Example 2: Reading output from INGRES and sending output to Visual Display Unit (VDU).

```
main()
{
        int from_ingres; /* File descriptor for reading end of P2 */
        wait_for_ingres(); /* Tell compiler 'wait_for_ingres'
                              is a routine */
        char c; /* c is of type character */
        .
        .
        if ((pid2 = fork()) == 0) wait_for_ingres();
         /* Create Child 2 and get it to listen to INGRES */
        .
        .
}
wait_for_ingres()
{
        read(from_ingres, &c,1); /* read one character at a time
```

```
        putchar(c);                from INGRES and output on VDU */
}
```

Filtering is achieved by C statements within the 'wait_for_ingres' routine which specify what is to be filtered. This is not shown in the above example but can be found in this routine (see Appendix 10).

Chapter 8 describes how the interface can be used and also gives an example on how the system could be modified to suit the varied needs of the user.

REFERENCES AND BIBLIOGRAPHY

[7.1] Bourne S.R., "The Unix System", Bell Laboratories, Addison-Wesley Publishing Company, 1983.

[7.2] Kernigham B.W., and Ritchie D.M., "The C Programming Language", Bell Laboratories, Prentice Hall, Inc., Englewood Cliffs, New Jersey 07632, 1978.

[7.3] Purdum J.J., Leslie T.C., Stegemoller A.L., "C Programmer's Library", Que Corporation, Indianapolis,

[7.4] Rochkind M.J., "Advanced Unix Programming", Prentice-Hall, Inc., Englee-wood Cliffs, New Jersey 07632, 1985.

[7.5] Stroustrup B., "The C++ Programming Language", AT & T Bell Laboratories, Addison-Wesley Publishing Company, New Jersey, 1986.

CHAPTER 8

APPLICATION OF THE INTERFACE

8.0 BACKGROUND

The purpose of a data base is to receive, retain, and provide information about a slice of reality. A data base should be looked upon as a tool and not an end by itself. Giving back all the information a database knows "by heart" is one aspect of the system. Another aspect of the system is to deduce an answer from other facts by means of programs. A third aspect is to store previously deduced information as if it had been entered from the external world.

These three mechanisms of the system can be best illustrated by thinking of the way a human being achieves answers to the following questions:

- What is 4 minus 3? --- Answer is achieved by "heart".
- What is 28 plus 42? --- Answer is achieved by deduction (computation).
- What is 42 plus 28 --- Answer is achieved by memorising the previous answer.

In fact these three mechanisms are organised as hierarchy similar to a memory hierarchy in a computer system:

First, see if you know the answer by heart, in case of failure try to deduce it by some appropriate rule (program) and in case of success keep the result for a while in case of a future identical question.

8.1 STRUCTURE OF SYSTEM

Figure 8.1 shows the structure of the interface. The system can be used in two ways. It can be used for information retrieval and as an associative process. These two functions are described in turn below.

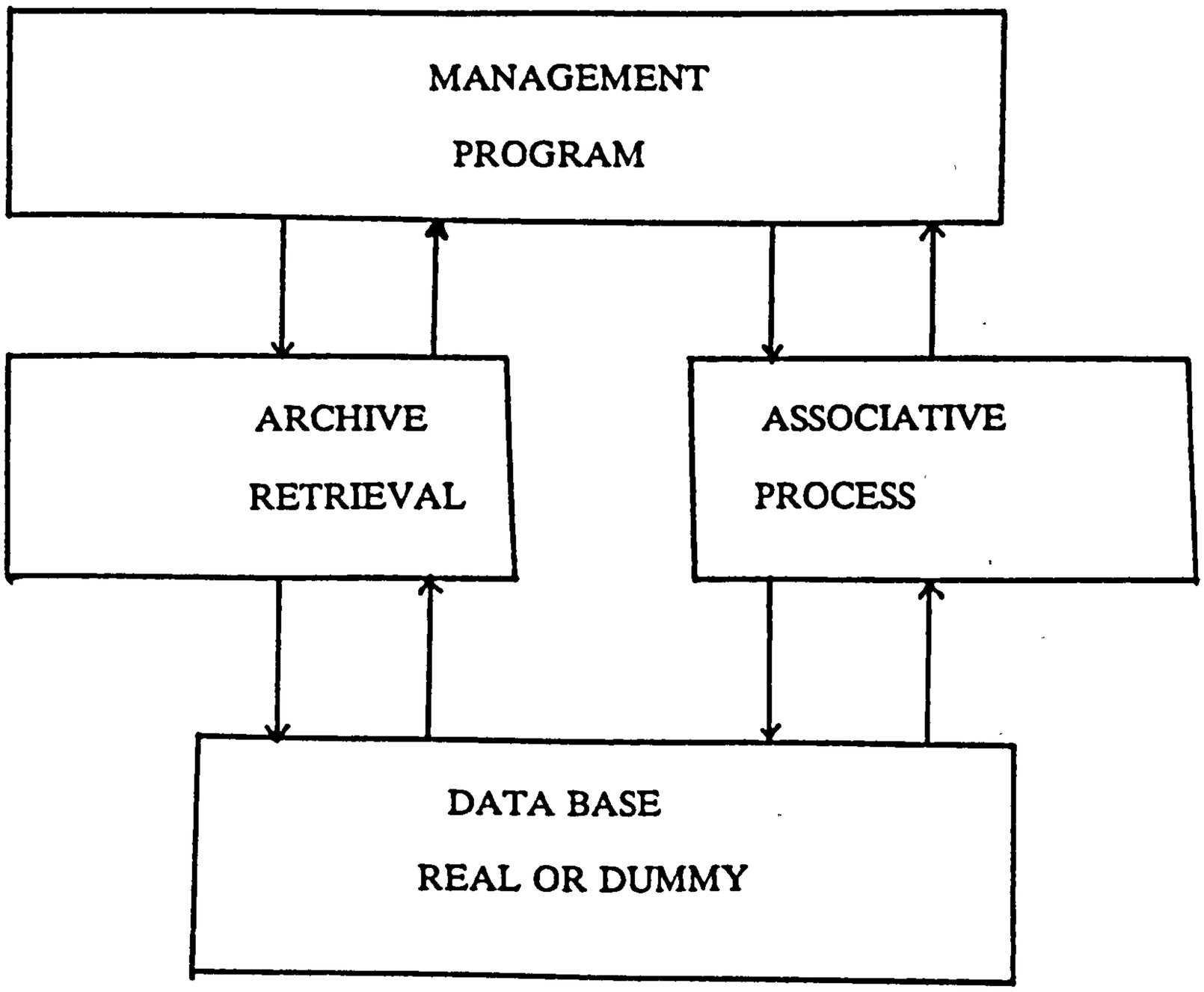


Figure 8.1 Structure of System

8.1.2 Information Retrieval

One may interrogate the data base and obtain information in an easier manner than by direct inquiry in the real world. This is further made easier by the user friendly interface which carries out a dialogue with the user, thereby guiding the user as to what information is available in the data base.

The user is thus not expected to have any knowledge of the INGRES query language (QUEL) as is the case in a direct interaction with the data base. The system builds up the query step by step as the user answers to simple questions asked by the interface. Also, qualifications to the query are built up in steps, thus enabling the user to gradually narrow their query to the desired one.

Another aspect of the system is that it allows the user to build queries from incomplete knowledge of the information asked for by the interface. This is achieved by the use of special character matching characteristics which are built into the data management system, reference [6.13]. For example, if the user is not sure of the exact detail of the information asked for by the interface, a few characters could be given to the system and that would be sufficient for the system to work out all the possible queries that satisfy that character combination. By adding qualifications the query could then be pruned down based on the information already supplied by the data base. Hence, a full knowledge of the data base is not required for successful application of the system.

Another aspect of the information retrieval system, is that it has been structured in such a way that further information could be added to the data base and further routines added to the interface to meet the added needs of the user without alteration of the basic structure of the system. This is achieved by the use of the "switch" multiple choice structure, (see switch, reference [6.20]).

Searching a data base is expensive with respect to search time. To eliminate repeated search for the same information, the system has been incorporated with a memory which stores previous queries, thus making it possible for the system to directly give back answers without searching the data base, to previously asked questions as illustrated in the example in Section 8.0.

8.1.2 Associative Process

This section of the system uses information already stored in the data base together with information supplied by the user to help the user make decisions. In order to be able to assess completely a particular bridge, the following parameters have been identified:

(1) category of rail or road on which bridge lies - the categorisation used was adopted from British Rail, Highway Department, and Scottish Development Department. Bridges on major trunk roads and on intercity lines are classified as Class A while those on side roads and railway branches are classified as Class B.

(2) archived data - this information is available from Authorities responsible for the maintenance of bridges, that is, British Rail, Highway Department and Scottish Development Department. These authorities were visited and the following records obtained:

- name and location of bridge on road or railway line;
- records of visual inspection carried out every four years dating back for the last twenty years;
- repairs done on the bridges over the past twenty years;
- statutory obligations attached to bridge;

(3) recent visual inspection results;

(4) non-destructive test results, if any;

(5) coring results if available;

(6) bridge dimensions.

Appendix 9 gives a listing of the contents of the current data base, that is, the information listed above. The routine 'decision_support' (see Appendix 10) demonstrates how

the above information can be utilised to assess a bridge. A full account on the MEXE method of assessment applied in this routine can be found in refences [3.4]

Appendix 9 shows the data that is currently available from the data base.

8.2 SYSTEM CHARACTERISTICS

The system has the following characteristics:

- the system is able to evolve - reality changes with time, hence the data base model will also change with time. Also, the needs of an enterprise change with time. The system was developed with this in mind and was developed in such a way as to enable modifications to be done with no change to the basic structure of the system. Section 8.4 outlines how the system could be modified to meet the added requirements of the user;
- efficiency as outlined in Section 8.3 below;
- the way of interrogating the system is independent of the mechanism it uses to get the answer. this quality is known as "semantic data independence".

8.3 SYSTEM PERFORMANCE

In INGRES, a user is allowed to execute a RETRIEVE INTO statement which creates a new relation and hence alters the relational schema, that is, the description of the database. This is analogous in general purpose programming language to allowing a user to create space for variables at run time. However, in programming languages, such variables are local to an invocation of the program and space disappears when the program terminates.

In INGRES, relations created during execution do not disappear when the program terminates (since the programmer may wish to use such relations again). In fact

INGRES keeps relations for a period of time specified by a database administrator and provides a SAVE command should the user wish that they be kept longer (see Appendix 6B).

This causes several problems. If the relation schema is to be altered at run time the following dilemmas occur:

(i) it may be impossible to do the alteration because the relation to be created at run time may depend on other relations (created by other routines) which do not yet exist in the schema (because the other routines have not yet been called upon).

(ii) there may be name conflicts; that is, a user may be required not to use the same name for a relation in different routines;

(iii) DESTROY relation_name is a legal command, and in this case the schema cannot be altered until all routines have been called upon;

(iv) it may be impossible to alter the schema because changes could easily depend on run time values.

Dilemmas (i) and (iv) were overcome by synchronisation of the three processes The INGRES monitor prints an asterisk ("*") at the beginning of each line to remind the user that INGRES is ready for the next input. Also, the monitor rings a bell after completing to process each query. By trapping these two messages, from INGRES, and using synchronisation signals, the second child and the parent process were forced to wait until INGRES had completed its tasks.

Dilemma (ii) was overcome by keeping in memory all created relations and automatically naming new relations with names that do not exist in the schema. Problem (iii) was avoided by destroying all relations that were created during execution of the program. This has the disadvantage of not allowing the user to keep relations for as long

as they require. However, the information may be kept on the UNIX level if required and transferred to INGRES using the COPY command (see Appendix 6B). This approach was taken to avoid repeated crushing of the system due to conflict in the names after several repeated usage of the system. Routines 'wait_for_ingres', 'wait_for_signal' and 'clean_up_ingres' (Appendix 10) carry out the above operations.

8.4 SYSTEM MODIFICATION

As already mentioned in Section 8.2, one of the characteristics of the system is being able to evolve. The following example describes how an additional relation could incorporate into the system. The example assumes that the relation has already been added to the data base.

In the following description, the line numbers refer to Appendix 10. Consider the case where the relation NEW_RELATION is to be incorporated into the system. The procedure with reference to Appendix 10 would be as follows:

- Under the section headed 'DECLARE DATA BASE BRIDGE'S DETAILS' (above line 358)
  - (i) NEW_RELATION would be added in routine 'rel_name' (line 358);
  - (ii) The routine 'NEW_RELATION_dom_name' specifying the domains of new_relation would be added as in the routine 'specify_dom_name' (line 367).
- Under the section headed 'DECLARE VARIABLES FOR VARIOUS SELECTIONS' (above line 404)
  - (i) The declaration "char NEW_RELATION_SELECTION" would be added (after line 410).
- In routine 'infor_retr' (line 416)

(i) mesg4 (lines 418 - 421) would be modified to include NEW_RELATION as one of the choices of relations available in the data base.

(ii) An additional case statement would be incorporated in the switch statement (after line 448) with the variables in brackets as in the other case statements in lines 441 - 448.

■ Immediately above the section 'EQUALITY OPERATORS' after line 573)

(i) the routine NEW_RELATION' would be added, taking 'specify' routine as an example, an exact copy of this routine would be created. The message statement in the routine should be altered to give domains of NEW_RELATION as well as 'n' in "mesgn" which should be greater than the maximum n already used as the system grows. This can easily be determined by using the editor (see reference [6.20]).

(ii) routine 'specify_selection' should be changed to

(iii) routine 'specify_dom_name' should be changed to 'NEW_RELATION_dom_name'

■ Under the section headed 'QUALIFICATIONS FOR INFORMATION RETRIEVAL' (above line 620)

(i) a case statement with 'NEW_RELATION_qual' routine should be added to the switch statement, (after line 639) with the variables in brackets as with the other case statements (lines 632 - 639).

■ Immediately above 'EQUALITY' (above line 853) should be placed

(i) the 'NEW_RELATION_qual' routine, (below line 852) also a copy of the 'specify_qual' routine.

(ii) in the printf statement, 'specifications' in the printf statement should be changed to 'NEW_RELATION' and the rest of the changes are as outlined above for the 'specify' routine.

In a similar manner, additions to domains can be accommodated by the system although this would seldom be done as it would result in loss of data already stored in the relation and should be done in exceptional cases only.

8.5 RUNNING THE SYSTEM

The system developed is currently available on the VAX (edee) machine in the Department of Electrical Engineering at the University of Edinburgh. The source file is called 'inter.c'. To use the system, the user must compile the program by the normal C compiler to produce an executable module. The command for this is

```
cc inter.c -lm
```

where the '-lm' flag specifies a special mathematics link-library which is used in the routine 'decision_support'. The executable module produced by the C compiler is used as the front end process to INGRES. The command for running the executable module is

```
a.out
```

which will get the system running. Chapter 9 demonstrates how the data base relating to bridges is used.

8.6 DATA BASE DESIGN FOR THE SYSTEM

The system will give optimum performance if the design of the data base is as described in Chapter 5, Section 5.4. However, the sytem will still operate even if the data base is not as designed as described in Chapter5. It is worth noting that the idea of normalisation should not restrict the user from creating relations that do not fall into the category of third normal form.

CHAPTER 9

DEMONSTRATION RUN

This chapter gives a demonstration run that would enable the user to use the system developed. The italics refer to the system prompts, while the bold face refer to the user responses. On typing a.out as already mentioned in Section 8.5 the initial prompt from the system would be:

Do you wish to obtain a list of available databases? (y/n): **y**

This enables the user to find out what data bases are available in INGRES at that particular time. This would vary as other users create their own data bases. A typical response would be:

demo hydrosite bridge

The data base demo gives a demonstration run on how one would use INGRES without the system. For the purposes of this thesis, the data base bridge is relevant. The next prompt from the system would be:

Enter name of database you wish to consult: **bridge**

This would invoke INGRES as described in Section 7.5. The system would then inform the user that data base bridge already exists. In the case where a user gives a data base name which does not exist, that data base would be created. The system then prompts the user with:

INGRES version 7.10 login

Tues Apr 2 15:01:09 1987

go

The first two lines include the INGRES version number (in this case version 7.1) and the current date. The "go" indicates that INGRES is ready for your interactions. This is followed by a print of the facilities offered by the system (menu), viz:

[I]NFORMATION RETRIEVAL

[D]ECISION SUPPORT SOFTWARE

[S]HOW RELATIONS IN A DATA BASE

[L]IST DOMAIN NAMES AND FORMATS

[P]RINT OUT CONTENTS OF A RELATION

[V]IEW QUERY BUFFER

[C]REATING AND MAINTAINING A DATA BASE USING INGRES

[Q]UIT INGRES

The facilities offered by the system will be demonstrated in turn. The following examples demonstrate how the facilities would be used.

9.1 SHOW RELATIONS IN A DATA BASE

A print out of the menu as above is followed by the dialogue:

Enter square bracketed letter for your selection.

Select: **s**

The following output would be obtained:

relation name	*relation owner*
attribute	*lfms*
maint	*lfms*
relation	*lfms*
indexes	*lfms*
ndt	*lfms*
specify	*lfms*
arch_dims	*lfms*
defects	*lfms*
integrities	*lfms*

type *lfms*

protect *lfms*

tree *lfms*

vi_inspect *lfms*

gvw_rest *lfms*

This shows what relations are in data base bridge. The relations ("attribute", "relation", "indexes", "integrities", "protect", and "tree") are INGRES built in system relations. This is the followed by the prompt:

Enter m for MAIN menu **m**

9.2 LIST DOMAIN NAMES AND FORMATS

The main menu as above is listed and the dialogue continues.

Enter square bracketed letter for your selection.

Select: **l**

This selection will offer information about a particular relation. Supposing we wished to know about the relation "vi_inspect", the dialogue would be:

Enter name of relation: **vi_inspect**

This would result in the following output:

Relation: *vi_inspect*

Owner: *lfms*

Tuple width: *60*

Saved until: *Tue Apr 22 16:36:48 1986*

Number of tuples: *9*

Storage structure: *paged heap*

Relation type: *user relation*

attribute name	*type*	*length*
name	*c*	*20*
vi_date	*c*	*10*
vi_details	*c*	*30*

This facility lists overall information about the "vi_inspect" relation together with each attribute, its format type and its length.

INGRES supports three data types: integer numbers, floating point numbers, and character strings. Character domains can be from 1 to 255 characters in length. Integer domains can be 1, 2, or 4 bytes in length. This means that integers can obtain a maximum value of 127; 32,767; and 2,147,483,647 respectively.

The maximum number of domains that a relation may have is 49. The number of tuples refers to the number of bridge information currently stored in the relation. The "tuple width" is the sum of the individual lengths of the attributes, that is, 20 + 10 + 30 = 60. The "heap" storage is as described in Section 6.5.1.

9.3 PRINT OUT CONTENTS OF A RELATION

After a listing of the main menu the dialogue continues as follows:

Enter square bracketed letter for your selection.

Select: **p**

Enter name of relation: **vi_inspect**

The above dialogue would yield:

vi_inspect relation

name	*vi_date*	*vi_details*
Gorgie	*16/09/1957*	*longitudinal cracks near arch edge*
Craiglockhart	*12/02/1959*	*bulging of spandrel walls*
Abbeyhill	*23/7/1961*	*depressions on bridge deck*
Easter Rd	*18/10/1963*	*cracking(longitudinal)_barrel*
Powderhall	*17/09/1965*	*cracking(diagonal), barrel*
Powderhall	*03/06/1967*	*bulging(spandrel walls)*
Portobello	*14/12/1969*	*arch out of shape*
Inveresk	*25/06/1971*	*lack of mortar at joints*
Newtongrange	*05/08/1973*	*cracks, spandrels at 1/4 points*

This facility enables the user to look up all domains of a relation. The domain names, "name", "vi_date", and "vi_details" give the name of the bridge, the date on which the visual inspection was carried out, and the details of the visual inspection respectively.

9.4 VIEW CONTENTS OF QUERY BUFFER

Having obtained a listing of the main menu, the dialogue is as follows:

Enter square bracketed letter for your selection.

Select: **v**

This facility is only used during the development of the system. It enables the system developer to check that the message sent to the query buffer by the system is as intended.

9.5 INFORMATION RETRIEVAL

Information can be retrieved from the data base as in the following dialogue:

Enter square bracketed letter for your selection.

Select: **i**

Which of the following do you require to consult?

1.Specification 2. Ndt 3.Maintenance 4. Visual Inspection

5. None of the above

Select by Number: **1 3**

This gives a listing of the relations that are currently supported by the information retrieval system. Additional relations can be added as need be as outlined in Section 8.4. A typical selection such as **1 3** would be followed by the prompt:

Of which of these on Specification do you require information?

1. Name 2. Location 3. Category 4. Route

Select by Number: **1 3**

This indicates the domains that are in the relation that offers the specification of the bridge and enables the user to choose those domains they require to obtain information on. A typical selection of **1 3** would result in the prompt:

Of which of the following on Maintenance do you wish to obtain information?

1.Name 2. Maintenance Date 3. Maintenance Details

Select by Number: **3**

This gives a list of the domains that are in a relation that offers information on the maintenance of the bridges.

The above dialogue implies that the user has requested for the name of the bridge, its category, and the details of the maintenance that has been carried out on the bridge. This is then followed by the prompt:

Do you wish to qualify your requests? (y/n): **n**

which enables the user to add qualifications to the request. The above response would result in the following output from the system:

requested0 relation

name	*category*	*mnt_details*
Abbeyhill	*rail*	*bolting of spandrels*
Craiglockhart	*rail*	*fill replaced with concrete*
Easter Rd	*rail*	*replacement of spandrels*
Gorgie	*rail*	*grouting on spandrels*
Inveresk	*rail*	
Newtongrange	*road*	
Portobello	*rail*	
Powderhall	*rail*	
Powderhall	*rail*	*replacement of arch*

The blanks in the maintenance details section indicate that no such records exist for the bridge in question.

In the case where the user wishes to add qualifications to their requests, that is, the response to the prompt:

Do you wish to qualify your requests? (y/n): **y**

is as above, that is, a "yes", the dialogue would proceed as follows:

Do you wish to qualify Specification? **y**

Do you wish to qualify Name? **n**

Do you wish to qualify Category? **y**

How many qualifications do you wish to place on this domain? **1**

Which one of these qualifications do you wish to apply?

1. Equal *4. Greater than or equal*

2. Not Equal *5. Less than*

3. Greater than *6. Less than or equal*

Select by Number: **1**

A typical selection such as **1**, that is, an equality qualification would result in:

Which one of the following pattern matching constructs

do you wish to use for your qualification?

1. Know the full qualification

2. Know the first characters of the qualification

3. Know the middle part of the qualification

4. Know only the last characters of the qualification

Select by Number: **1**

In the case where the user does not know the full qualification, choices 2, 3, or 4 can be used to make up the qualification as appropriate which would yield:

Please enter your qualification. **rail**

If the user does not know the full qualification and made choice **2** from above, the qualification "ra" would still suffice. The dialogue continues as follows:

Do you wish to qualify Maintenance? **y**

Do you wish to qualify mnt_details? **y**

How many qualifications do you wish to place on this domain? **1**

Which one of these logical operators do you wish to apply?

1. And

2. Or

3. Not

Select by Number: **1**

Which one of these qualifications do you wish to apply?

1. Equal *4. Greater than or equal*

2. Not Equal *5. Less than*

3. Greater than *6. Less than or equal*

Select by Number: **1**

Which one of the following pattern matching constructs do you wish to use for your qualification?

1. Know the full qualification

2. Know the first characters of the qualification

3. Know the middle part of the qualification

4. Know only the last characters of the qualification

Select by Number: **1**

Please enter your qualification. **rail**

The above interrogation implies that the user wishes to obtain information about bridges that belong to the category "rail" and have maintenance details beginning with the letters "bolt". The following output is obtained:

requested1 relation

name	*category*	*mnt_details*
Abbeyhill	*rail*	*bolting of spandrels*

9.6 DECISION SUPPORT SOFTWARE

This section demonstrates how the information in the data base can be used to determine the load carrying capacity of a single span arch of span less than 18 meters in terms of either full Construction and Use (C&U) loading or specified gross vehicle weights. After a listing of the main menu, the dialogue is as follows:

Enter square bracketed letter for your selection.

Select: **d**

Please enter name of bridge you wish to assess: **Gorgie**

Enter location of bridge: **Slateford**

The relation "arch_dims" (arch dimensions) has all the dimensions shown in Appendix 3 (Figure 8.1), which are required for the computation of the various factors that are required for the assessment of a structure by the MEXE method. The system will look up the thickness of the barrel from the relation "arch_dims" and the following message will be printed:

Thickness of barrel is 0.04 metres

The system then lists the details of the construction of the joints viz:

1. *Unpointed joints, pointing in poor condition and joints with up to 12mm from the edge insufficiently filled.*
2. *Joints with from 12mm to one tenth of the thickness of the barrel insufficiently filled.*
3. *Joints insufficiently filled for more than one-tenth the thickness of the barrel.*
4. *Pointed joints in good condition*

Select by number the construction of joint: **1**

The system will provide the depth factor that corresponds to the choice of construction of the joint. The factors are in Appendix 3, Table 3.5. In the case where the selection above is "3", the system will ask for an estimate of the depth of missing mortar, and the arch barrel thickness will be reduced by this amount. This is then followed by the request for information regarding the barrel details.

1. *Granite and Whitstone whether random or coursed and all built-in-course masonry except limestone, all with large*

shaped voussoirs.

2. *Concrete or engineering bricks and similar sized masonry (not limestone).*
3. *Limestone, whether random or coursed, good random masonry and building bricks, all in good condition.*
4. *Masonry of any kind in poor condition (many voussoirs flaking or badly spalling, shearing etc.). Some discretion is permitted if the dilapidation is only moderate.*

Select by number the arch barrel details: **1**

The system then looks up the barrel factor that corresponds to the above choice (see Appendix 3, Table 3.1) and then provides the user with the possible fill materials, viz:

1. *Concrete*
2. *Grouted materials (other than those with a clay content)*
3. *Well compacted materials*
4. *Weak materials evidenced by tracking of the carriageway surface.*

Select by number the filling: **4**

The fill factor corresponding to the above choice is provided by the software (see Appendix 3, Table 3.2). and the system offers information on the widths of the joints, viz:

1. *Joints with widths up to 6mm*
2. *Joints with widths between 6mm and 12mm*
3. *Joints with widths over 12mm*

Select by number the width of joints: **3**

The width factor corresponding to the chosen width of joint is then provided by the software (see Appendix 3, Table 3.3) and the condition of joint is requested by:

1. Mortar in good condition

2. Loose or friable mortar

Select by number the condition of joint: **2**

The mortar factor corresponding to the chosen joint condition is provided by the system (see Appendix 3, Table 3.4). Finally the dialogue would be as follows:

Enter condition factor of bridge (0 -1.0): **0.3**

Guidance on the choice of condition factor is given in reference [3.5] The system will prompt the user with the following messages:

Immediate consideration should be given to the repair or reconstruction of the bridge.

Modified Axle Load = 5.5 tonnes

Maximum Gross Vehicle Weight = 32.5 tonnes

This is for a HGV (Heavy Goods Vehicle) with 4 axles

The routine axle_lift_off computes the Axle factors and then finds the maximum gross weight of the C&U vehicles from the relation "gvw_rest" (gross vehicle weight restrictions for masonry structures).

9.7 CREATING AND MAINTAING A DATA BASE USING INGRES

This section demonstrates how to create, structure and maintain relations in INGRES using the software. On obtaining a listing of the main menu, the dialogue would be as follows:

Enter square bracketed letter for your selection.

Select: **c**

This would then be followed by a listing of the facilities available for creating and maintaining relations. The sub-menu listing would be as follows:

[C]reate an empty relation

[F]orming a relation from existing relations

[D]estroy a relation

[E]rase contents of a relation

[H]ow to copy whole relations to INGRES

[M]odify system relations

[T]o destroy a data base

[S]torage Structures in INGRES

[Q]uit sub menu to MAIN MENU

9.7.1 Create an Empty Relation

Suppose we wished to create the relation "donation" with attributes: name, amount and ext (extension); where the attributes refer to the name of the donor, the amount they donated and their telephone extension number respectively. A listing of the sub-menu would be followed by the dialogue:

Enter square bracketed letter for your selection.

Select: **c**

Enter name of relation: **donation**

Enter number of domains: **3**

Enter domain name: **name**

Enter format type: **c15**

Enter domain name: **amount**

Enter format type: **f4**

Enter domain name: **ext**

Enter format type: **i2**

The format types are as outlined in Section 3 of Appendix 6B. The above dialogue results in the creation of the relation "donation" with no tuples is it.

9.7.2 Forming a Relation From Existing Relations

This facility enables the user to form a new relation from one or more existing relations. Suppose the user wishes to create a new relation, "new_relation", with attributes, name (name of bridge), location (bridge location), and vi_details (visual inspection details). The first two attributes are in "specify" relation while the third attribute is in "vi_inspect" relation (see Appendix 9). The dialogue after a listing of the sub-menu would be as follows:

Enter square bracketed letter for your selection.

Select: **f**

*Enter name of new relation:***new_relation**

Enter number of source relations: **2**

Enter name of source relation : **specify**

Enter name of source relation : **vi_inspect**

Enter number of domains to be retrieved: **3**

Enter domain name: **name**

Enter domain name's relation: **specify**

Enter domain name: **location**

Enter domain name's relation: **specify**

Enter domain name: **vi_details**

Enter domain name's relation: **vi_inspect**

Enter name of domain common to all relations: **name**

9.7.3 Destroy a Relation

This facility removes relations from the data base, and removes constraints or permissions from a relation. Only the relation owner may destroy a relation or its permissions and integrity constraints. Suppose the user wishes to destroy the rela-

tion "donation", created above. The procedure after a listing of the sub-menu would be as follows:

Enter square bracketed letter for your selection.

Select: **d**

Enter name of database you wish to destroy: **donation**

9.7.4 Erase Contents of a relation

This facility enables the user to delete tuples from a relation. The user must be the owner of the relation. Consider the case where the user wishes to delete the tuples in the relation "new_relation". The procedure after a listing of the sub-menu would be as follows:

Enter square bracketed letter for your selection.

Select: **e**

Enter name of relation: **new_relation**

9.7.5 How to Copy Whole Relations to INGRES

This facility may be used to move data between standard UNIX files and INGRES. This is used when data is being entered into a relation for the first time. To use this facility, the user must first create a UNIX file (typically using "ed", see Section 6, Appendix 6B) containing the data. For example, to create data for the "donation" relation using the editor, would be as follows:

```
% ed newdom
a
bill,3.50,302
susan,,100
w
```

q

%

The "%" sign indicates the UNIX shell prompt. To use this facility, the procedure would be as follows:

Enter square bracketed letter for your selection.

Select: **h**

Enter name of file: **newdom**

Enter name of relation: **donation**

Enter number of domains: **3**

Enter domain name: **name**

Enter domain name: **amount**

Enter domain name: **ext**

The relation "donation" would now look like:

name	amount	ext
bill	3.500	302
susan		100

9.7.6 Modify System Relations

This facility is as outlined in Section 5 of Appendix 6B. The procedure after a listing of the sub-menu is as follows:

Enter square bracketed letter for your selection.

Select: **m**

Enter name of data base: **bridge**

modifying relation

modifying attribute

modifying indexes

modifying tree

modifying protect

modifying integrities

The system relations are modified to gain maximum access performance when running INGRES.

9.7.7 To Destroy a data base

This facility is as outlined in Section 3 of Appendix 6B. The procedure after a listing of the sub-menu would be as follows:

Enter square bracketed letter for your selection.

Select: **t**

Enter name of data base you wish to destroy: **junk**

Data base junk has been used for demonstration purposes, as the other data bases are of use.

9.7.8 Storage Structures in INGRES

This facility is as outlined is Section 6.5. A fill factor is used to specify how full to make each primary page. This decision should be based on whether more tuples will be appended to the relation. For example, a fill factor of 25 will leave each page 25% full, or in other words 75% empty. The dialogue after a listing of the sub-menu would proceed as follows:

Enter square bracketed letter for your selection.

Select: **s**

Do you wish to know what relations are in

the data base? (y/n): **n**

Enter name of relation whose storage structure

you wish to modify: **donation**

Which one of the following storage structures do yo wish

to apply to the relation:

1. Hash 2. Isam 3. Heap

Select by Number **1**

Do you wish to know what domains are in the relation? (y/n): **n**

Enter name of domain whose storage structure you

wish to modify: **name**

Do you wish to specify a fill factor? (y/n): **y**

Enter fill factor (1 - 100): **50**

9.7.9 Quit Sub Menu to Main Menu

This will result in the following prompt:

[Enter 'm' for MAIN MENU]

which will result in a listing of the main menu. The facility QUIT INGRES, terminates the software.

The system will also prompt the user with the message:

Do you wish to obtain a copy of your request? (y/n):

after every query has been processed. This enables the user to choose the information that they would require printed on paper. This information is stored in a file called "hard_copy". The user must then send this file for printing to a relevant printer. This file is deleted each time the system is run, that is each time "a.out" is run. Hence, the user must send for a print out before the next rerun of the system.

CHAPTER 10

DISCUSSION

10.0 CONCLUDING REMARKS

At the present moment, it would be difficult to subject the arguments in favour of bridge inspection to a quantitative economic analysis. There are considerable differences between the existing types of structure, the materials used and the construction regulations applied when building the bridges. In addition the "history" of bridge inspection still covers too short a period for it to be feasible to make a systematic statistical analysis, comparing the behaviour of bridges which have not been inspected with that of bridges which have been subjected to periodic inspection and assessment.

Bridge documentation should be considered as indispensable for assessing bridge structures for managerial and engineering purposes. It provides all the information needed for making decisions on engineering, economic and policy matters.

Engineering aspects are mainly linked with the technicalities of assuring safety and serviceability of bridge structures.

Against this background, the requirements for appropriate bridge documentation can be identified as:

- systematic collection and filing of relevant data, information, records and documents;
- continuous updating during the total actual life of the structure;
- retrieving of data and information relevant to the actual form and level of organisation and to the scale of the problem.

These requirements are best met by the use of a data system such as described in this thesis.

Considerations of the available information should lead to one of the following conclusions:

- the bridge is adequate for current use for its normal life provided
- that it is maintained properly
- the bridge, although adequate at present, may not remain so in future
- the bridge is inadequate for the current use but may or may not be adequate for alternative uses
- the bridge is inadequate and needs remedial measures
- the bridge is unsafe and beyond normal repair
- the information is not sufficient to reach a definite conclusion.

The information stored in the data base is made easily available to the Engineer by the information retrieval software, thereby enabling the above conclusions to be made with greater ease than would be the case when information is not easily available. For example, by looking at the maintenance records, it would be possible to decide if the bridge is properly maintained. Also, by using the decision support software (see Section 9.6), together with the information stored in the data base, it is possible to determine the maximum gross weight of the C&U vehicles which the arch can carry. The C & U regulations are in relation C&U_Reg, Appendix 8.

The advantages of the system over a conventional system are that it highlights those parts of the data that are essential in the decision making process (see Section 9.5). Furthermore, a conventional system only retrieves data from the data base and can not make use of it after the retrieval process. On the other hand the system retrieves data and can make use it as is the case with the determination of the maximum gross weigth

of the C&U vehicles using the decision support sofware.

The remedial measures that could be carried out on masonry bridges are as outlined in Section 3.3. These may be summarised as; grouting, pointing and major overhall.

10.1 IMPLICATIONS TO CIVIL ENGINEERS

Bridge inspection is not an end in itself. It should be seen as an essential element of the ov all system comprising design, construction, inspection, bridge classification, operation, maintenance, repair, reconstruction, research and related specifications.

The reasons for bridge inspection have been outlined in Chapter 1 and refer broadly speaking, to ensuring safety, conserving initial investments, and responding to a number of complementary (often not less important) requirements and trends.

A well established inspection data system will provide the necessary information for the control of traffic using the structure, especially exceptional vehicles or convoys. The routine 'decision_support' (Appendix 10) demonstrates how this is achieved.

Detection of defects and deterioration set the needs for maintenance, repair or replacement. The identification of early signs of distress can considerably reduce the total cost of the remedial measures to be taken.

Results from inspection may form the basis for future research with the objective of analysing the observations of damage which have not been fully explained, and of enhancing knowledge on the structural behaviour of the bridges. In particular inspection data highlighting defects that are common to a specific type of structure, will show the need for in depth studies. For example, there have been cases that observations made on a single bridge have drawn the attention to defects which so far had not been detected but were relevant to a whole set of bridge types, reference [3.2].

10.2 DISCUSSION

The associative process of the system could not be fully developed at this stage due to the fact that most of the project time was spent developing the interface between the user and INGRES. Furthermore, for a successful development of the associative process, it is essential that a comprehensive data base be developed initially. This can be best achieved by a feed back from the practising engineer.

Initially, a specific interface for bridge inspection was developed. This was abandoned on the realisation that a specific interface requires a well defined data base. Hence the approach for specific but generalised interface was adopted.

Because of its flexibility, the system can be easily used by different departments until a fully fleshed data base has been developed. With a well documented data base it would then be possible, using the information retrieval process, to identify common modes of failures as well as their relationships to such factors as geographical location, soil type and traffic patterns. This would be an efficient way of categorising the bridges according to common defects and such data as those bridges located in positions where there is likelihood of damage due to impact from vehicles.

The development of distributed data bases, that is data bases that can communicate, makes this project potentially viable to large organisations such as British Rail and Highways Department as these organisations would be able to share data from various regions.

A considerable amount of research is required in the improvement of the assessment techniques before they can be of any practical use to the assessment process. However, with the use of data bases, it is possible to improve the standards of assessment using the idea of fuzzy logic for the decision making process. Fuzzy logic enables decisions to be arrived at with incomplete data available which is usually the case with real life

problems and far much so with masonry bridges.

This type of work can only be carried out when a fully fleshed data base is available. For this project data was collected from British Rail, Highway Department, and Scottish Development Department in Edinburgh only. This data was then used as a guideline to the type of data that is available. In future the development of an expert system would be more attractive than a data base management system. The expert system approach allows the use of fuzzy logic in the decision software development.

In developing the associative process, in some cases, dummy data was employed to enable the development of the system. However, most of the data used (95%) was real data. Appendix 8 shows the type of data that is currently available in the data base "bridge".

The user can by adding suitable qualifications to the information retrieval process obtain any relationships between for example types of defects and the location of the bridge.

APPENDIX 3: MEXE METHOD

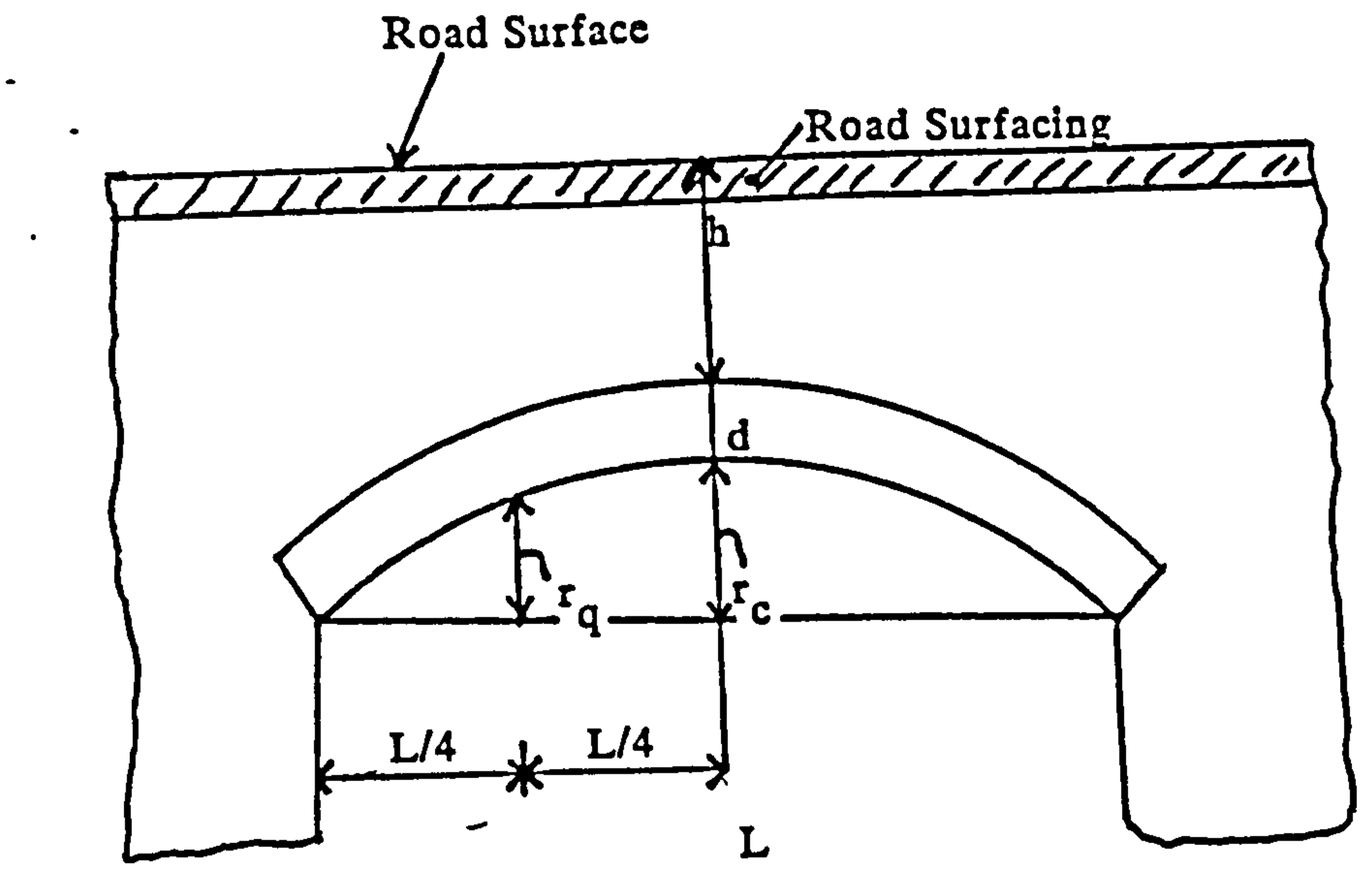


Figure 8.1 Arch Dimensions

Figure 8.1 shows the dimensions of the arch that are required for the assessment of a masonry arch bridge by the modified MEXE method which is used in routine decision support (see Appendix 10). The provisional axle loading (PAL) is given by the expression:

$$PAL = \frac{740(d+h)^2}{L^{1.3}}$$

The provisional axle load is then modified by various factors to allow for the shape of the arch, construction materials, dimensions of the arch barrel and any defects (see reference 3.4).

MODIFYING FACTORS

Span/Rise Factor

The strength of flat arches under a given loading is less than that of arches with a steeper profile, hence the provisional assessment need to be adjusted to take this into account. Figure 3 gives the appropriate span/rise factor for the different ratios (from reference [3.4]).

Profile Factor

Elliptical arches are not as strong as segmental and parabolic arches of similar span/rise ratio and barrel thickness, reference [3.4]. Adjustment is made to allow for a profile other than the standard parabolic, for which $r_q/r_c = 3/4$. The profile factor for ratios r_q/r_c less than or equal to 0.75 is taken to be unity, and for ratios greater than 0.75 is calculated from the expression:

$$F_p = 2.3\left[\frac{r_c - r_q}{r_c}\right]^{0.6}$$

Material Factor

The material factor (F_m) is obtained from the following formula:

$$F_m = \frac{(F_b.d)+(F_f.h)}{d+h}$$

where the barrel factor (F_b) and the fill factor (F_f) are as in Tables 3.1 and 3.2 (taken from refence 3.4).

Joint Factor

The strength and stability of the arch barrel depend, to a large extent, on the size and

condition of the joints. The joint factor (F_j) is obtained from the following formula:

$$F_j = F_w . F_d . F_{mo}$$

where the width factor (F_w) and the mortar factor (F_{mo}) are obtained from Tables 3.3 and 3.4 respectively (taken from reference 3.4). The depth factor (F_d) is obtained from Table 3.5.

Condition Factor

The estimation of the preceding factors is based on quantitative information obtained from a close inspection of the structure. The factor for the condition of the bridge depends much more on an objective assessment of the importance of the various cracks and deformations (see Chapter 2) which may be present and how far they may be counter-balanced by indications of good material and workmanship. Guidance on the choice of condition factor is given in references [3.3 and3.4]. a condition factor of 0.4 or less implies that the bridge should be rehabilitated immediately).

Modified Axle Load

The span/rise profile, material, joint and condition factors should be applied together with the provisional axle loading obtained above in order to determine the modified axle load (tonnes) which represents the allowable loading on the arch from a double axled bogie configuration with no 'lift-off' from any axle. The modified axle load is calculated from the following equation:

$$MAL = F_{sr} . F_p . F_m . F_j . F_c . PAL$$

Axle lift-off

The lift-off case relates to circumstances when an axle of double or triple axled bogie can loose contact, either partially or completely, with the road surface and transfer

some of its load to the other axles in the bogie. A lift-off case results from the presence of any of the following conditions:

- vertical road alignment with a small radius of curvature e.g. a humped back bridge.
- arch located at the bottom of a hill or on a straight length of road where approach speeds are likely to be high..
- irregularities in the road surface.

The unrounded value of the midified axle load is then multipled by the appropriate axle factors Figure 3.3 (reference 3.4) to give the allowable axle loads for single and multiple axles.

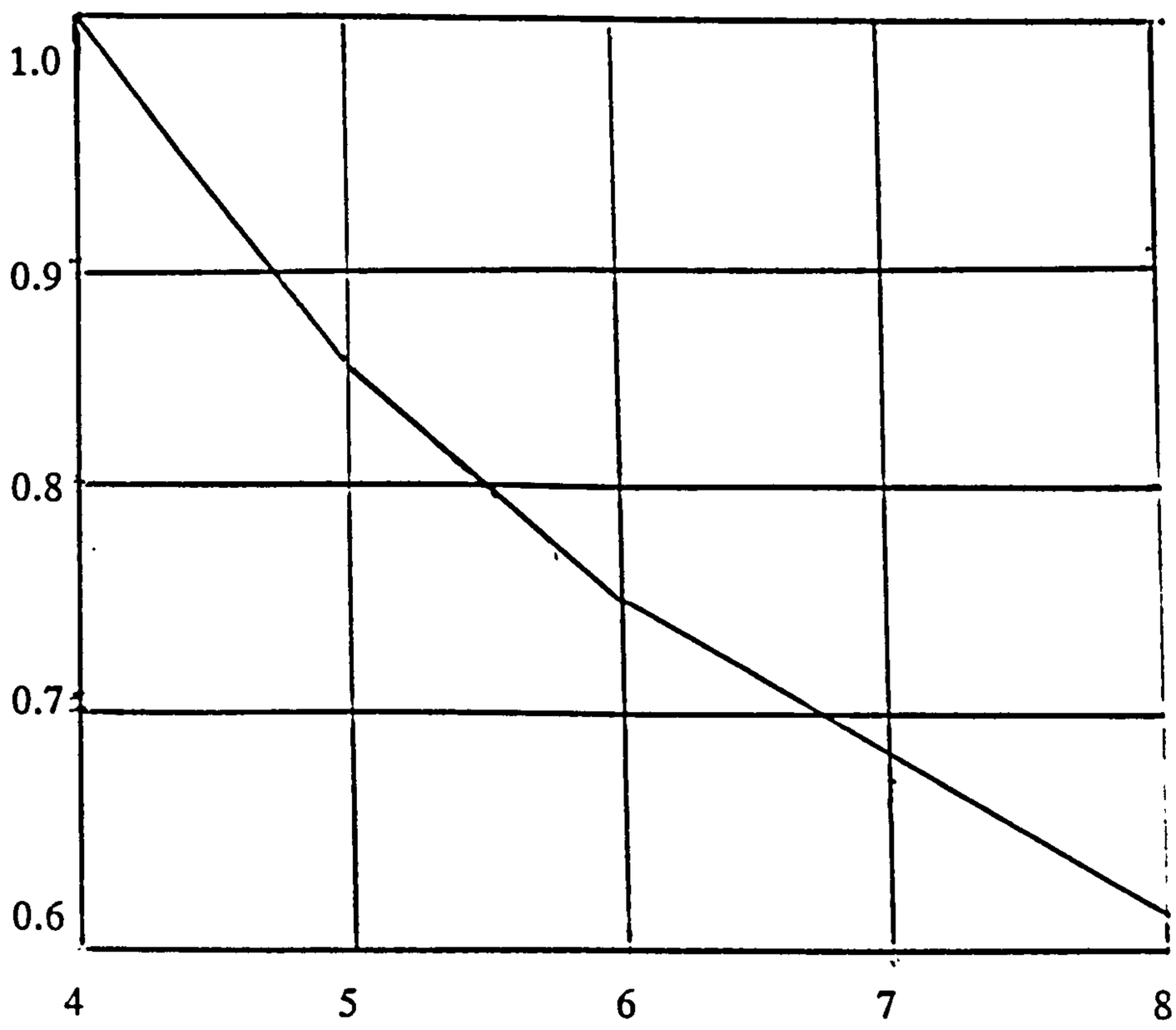


Figure 3 Span/Rise Factor

Arch Barrel	Barrel Factor (F_b)
Granite and Whitstone whether random or coursed and all built-in-course masonry except limestone limestone, all with large shaped voussoirs.	1.5
Concrete or engineering bricks and similar sized masonry (not limestone).	1.2
Limestone, whether random or coursed, good random masonry and building bricks, all in good conditon.	1.0
Masonry of any kind in poor conditon (many voussoirs flaking or badly spalling, shearing etc.). Some discretion is permitted if the dilapidation is only moderate.	0.7

Table 3.1 Barrel Factor

Filling	Fill Factor (F_f)
Concrete	1.0
Grouted materials (other than those with a clay content)	0.9
Well compacted materials	0.7
Weak materilas evidenced by tracking of the carriageway surface.	0.5

Table 3.2 Fill Factor

Width of Joint	Width Factor (F_w)
Joints with widths up to 6mm	1.0
Joints with widths between 6mm and 12mm	0.9
Joints with widths over 12mm	0.8

Table 3.3 Width Factor

Condition of Joint	Mortar Factor (F_{mo})
Mortar in good condition	1.0
Loose or friable mortar	0.9

Table 3.4 Mortar Factor

Construction of Joint	Depth Factor (F_d)
Unpointed joints, pointing in poor condition and joints with up to 12.5mm from the edge insufficiently filled.	0.9
Joints with from 12.5mm to one tenth of the thickness of the barrel insufficiently filled.	0.8
Joints insufficiently filled for more than one-tenth the thickness of the barrel.	At the Engineer's discretion
Pointed joints in good condition	1.0

Table 3.5 Depth Factor

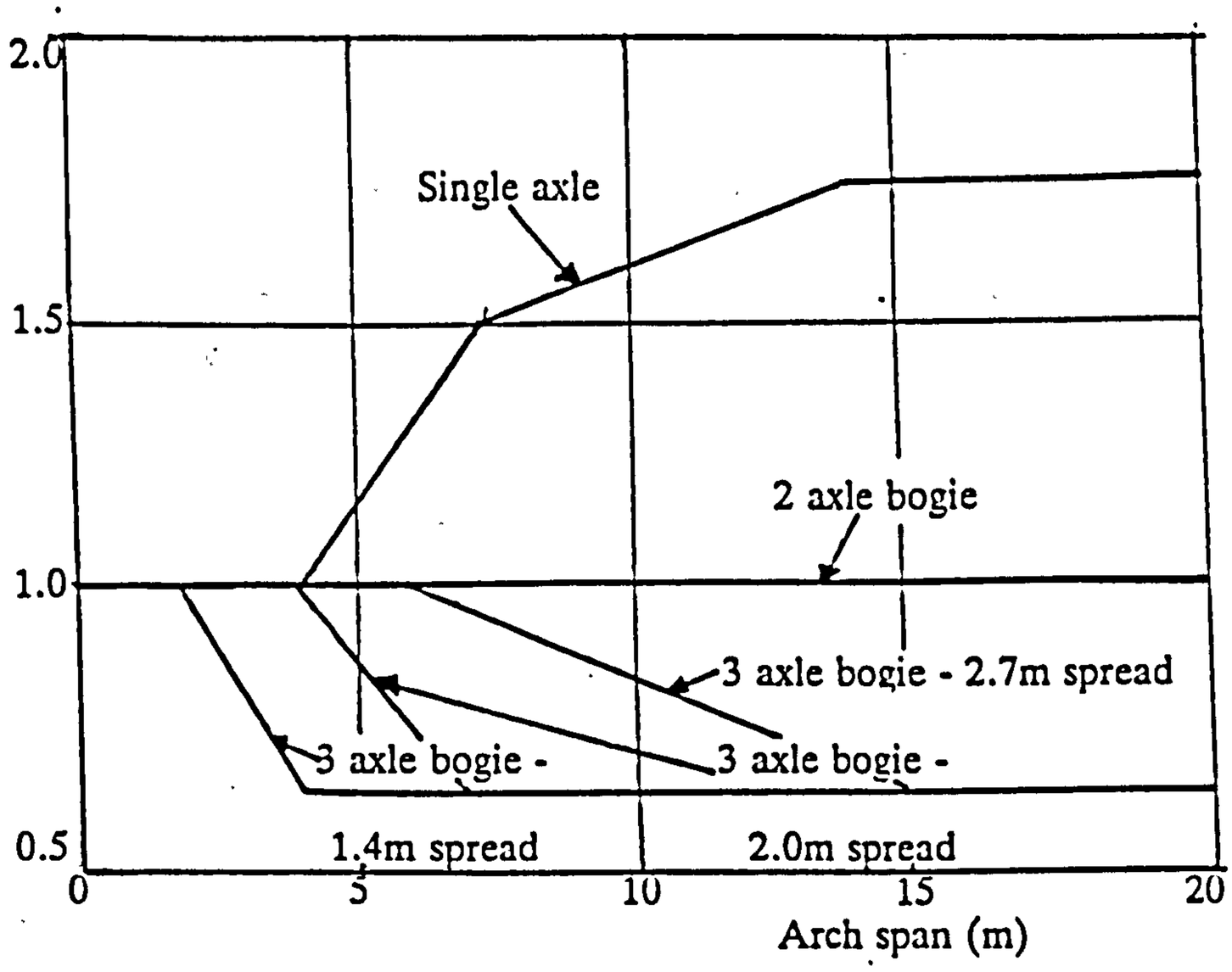


Figure 3.2 (a) No Axle Lift-Off

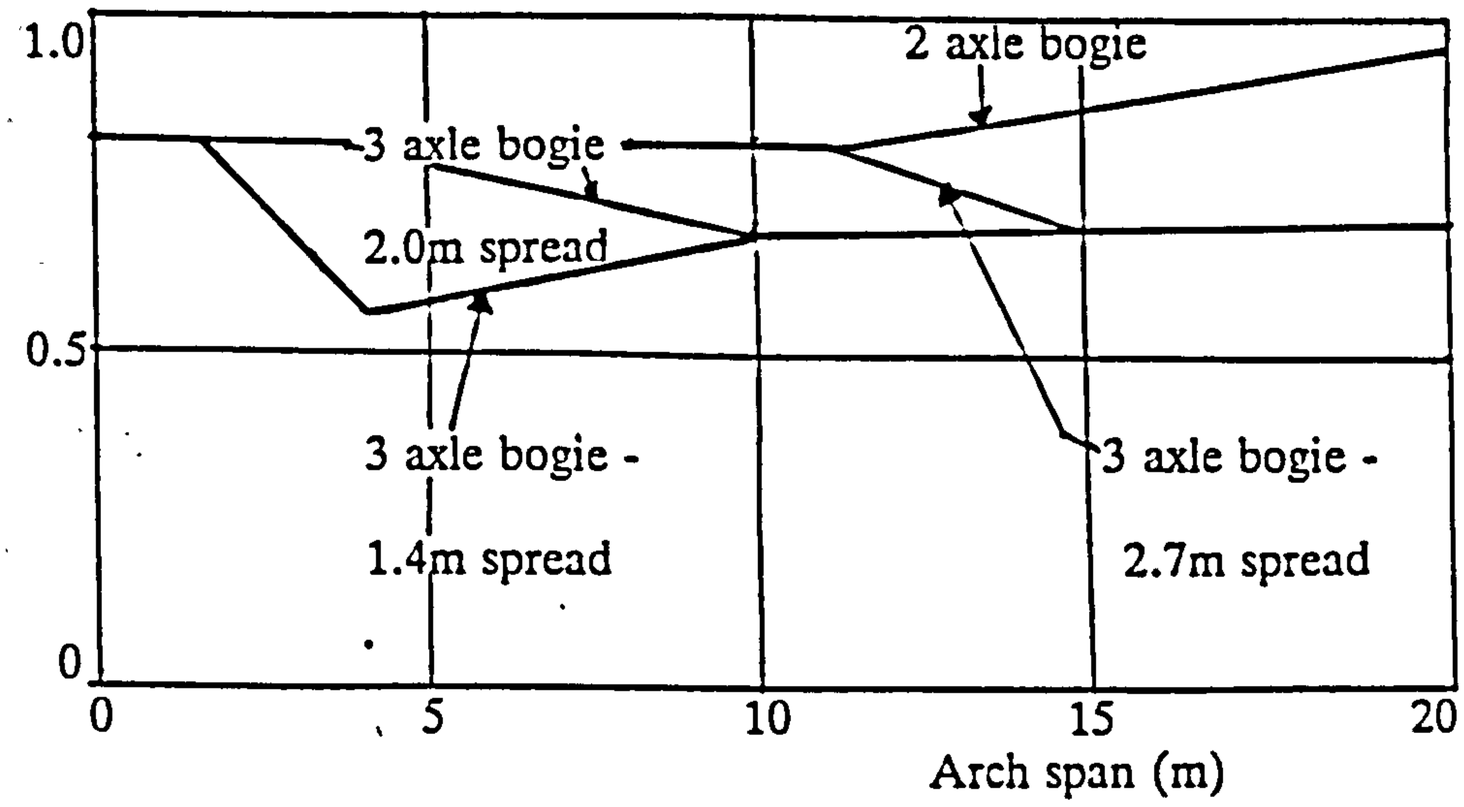


Figure 3.2 (b) With Axle Lift-Off

Figure 3.3 Conversion of Modified Axle Loads to Single, Double, and Triple Axles.

APPENDIX 6A: QUEL COMMANDS

STUDENT

NAME	DEPT	YEAR	DOS	AGE
Bloggs	History	2	Carlson	20
Smith	Geography	4	Wilson	24
Judy	Engineering	3	Hardy	20
Jones	History	1	Betty	23

Figure 6A.1 STUDENT Relation

Indicated above is a student relation with domains NAME, DEPT, YEAR, DOS and AGE. Each student has a name, belongs to a department, is attending some particular year of the course, has a Director of Studies(DOS) and has an age. The QUEL statements that follow suggest some valid QUEL statements and are based on this particular relation.

Example 1: Finding the birth date of Judy.

```
RANGE OF S IS STUDENT

RETRIEVE INTO W (BDATE = 1986 - S.AGE)
    WHERE S.NAME = "Jones"
```

In this case, S is a tuple variable which ranges over the STUDENT relation and may be thought of as a marker which moves down the STUDENT relation. S, by itself, refers to the STUDENT relation while S.NAME refers to the NAME domain of the STUDENT relation.

All tuples in the relation are found which satisfy the qualification

has a single domain, BDATE that has been calculated for each tuple. If the result rela-

tion is omitted, qualifying tuples are returned to the calling process. If this process is the terminal monitor, it in turn prints them on the user's terminal. Other front end processes may do what they wish with such tuples.

Furthermore, in the target-list, the "result-domain" may be omitted if function is of the form Variable.Attribute (for example, NAME = S.NAME may be written as S.NAME - see Example 5. Note also that Jones must be in quotes ("Jones"). The only way INGRES will recognise character strings (for example words) is to enclose them in quotes.

Example 2: Deleting the information about student Bloggs.

```
RANGE OF S is STUDENT

DELETE S WHERE S.NAME = "Bloggs"
```

In this example, all tuples corresponding to all students named Bloggs are deleted from the relation.

Example 3: Increasing the age of student Judy by 10 percent.

```
RANGE OF S IS STUDENT

REPLACE S(AGE BY 1.1 * S.AGE)
    WHERE S.NAME = "Judy"
```

In this case S.AGE is to be replaced by 1.1 * S.AGE for the tuples where S.AGE = "Judy". (Also, note that the keywords IS and BY may be used interchangeably with "=" in any QUEL statement, which improves the readability of the query).

INGRES supports arithmetic operators such as multiplication (*), subtraction and unary negation (-), as well as aggregation operators which include as ABS (absolute value), MAX (maximum) and AVG (average). Also INGRES supports equality operators such as greater than (>), equal to (=). A brief but complete description of

what is supported by INGRES can be found in [6.13].

Examples 4 and 5 demonstrate the use of some of the aggregation operators.

Example 4: Replacing the age of all History Department students by the average History Department age.

```
RANGE OF S IS student
REPLACE S (AGE BY AVG(S.AGE WHILE S.DEPT = "History"))
    WHERE S.DEPT = "History"
```

The average (AVG) is to be taken of the AGE attribute for those tuples satisfying the qualifications S.DEPT = "History". Note that AVG (S.AGE WHERE S.DEPT = "History") is a scalar valued and consequently will be called an aggregate. More general aggregates are possible as suggested by examples.

Example 5: Finding those Departments whose average age exceeds the University-wide average age, both averages to be taken only for those students whose age exceeds 23.

```
RANGE OF S IS STUDENT
RETRIEVE INTO HIGHAGE(S.DEPT)
WHERE AVG(S.AGE BY S.DEPT WHERE S.AGE > 23)
    >
    AVG(S.AGE WHERE S.AGE > 23)
```

Here, AVG(S.AGE BY S.DEPT WHERE S.AGE > 23) is an aggregate function and takes a value of S.DEPT. This value is the aggregate AVG (S.AGE WHERE S.AGE > 23 and S.DEPT = value).

The qualification expression for the statements is then true for departments for which this aggregate function exceeds the aggregate AVG(S.AGE WHERE S.AGE > 23).

APPENDIX 6B: INGRES UTILITY COMMANDS

In addition to the QUEL commands described in Appendix 6A, INGRES supports a variety of utility commands which can be classified into six major categories.

1) Invocation of INGRES:

INGRES can be invoked by executing from UNIX the command

INGRES database-name

This command executed from UNIX "logs in" a user to a specified database. Thereafter, the user may issue all other commands (except those executed directly from UNIX) within the environment of the invoked database.

2) Creation and Destruction of Databases

A database can be created or destroyed only from the UNIX level. The command for creating a database is

CREATDB database-name

and the command for destroying the database is

DESTROYDB database-name

The invoker of CREATDB must be authorised to create databases by the super-user and automatically becomes the database administrator with the authority to destroy the database.

3) Creating and destroying relations.

INGRES supports two ways of creating relations,

CREATE relation-name (domain-name IS format

RETRIEVE INTO [result-name](target-list)

[WHERE Qualification]

CREATE is used to create a new relation with no tuples in it. RETRIEVE INTO is used to form a new relation from one or more existing relations. A relation can be destroyed by the command

DESTROY relation-name

These commands create and destroy relations within the current database. The invoker of the the CREATE (or RETRIEVE INTO) command becomes the owner of the relation created with the power to destroy the created relation.

The current format types accepted by INGRES are:

- i1, i2, i4 (1, 2 and 4 byte integers);
- f4, f8 (4 and 8 byte floating point numbers);
- c1,c2,....c255 (1, 2,.....255 byte fixed length ASCII character strings.

4) Copying Data To and From INGRES

The command for copying data to and from INGRES is of the general form

COPY relation-name (domain-name IS format,
domain-name IS format,....)

direction "present working directory filename"

COPY transfers an entire relation to or from a UNIX file whose name is "filename". Direction is either TO or FROM. The format for each domain is a description of how it appears (or is to appear) in the UNIX file. The relation relation-name must exist and have domain names identical to the ones appearing in the COPY command. However, the formats need not necessarily be identical, and copy will automatically convert data types. The copy command also supports dummy and variable length fields in a UNIX

file. A stylised version of COPY is the PRINT command

PRINT relation-name

PRINT copies a relation onto the user's terminal, formatting it as a report.

5) Storage Structure Modification

The relation created by the user can be converted to any of the storage structures supported by INGRES using the MODIFY command

MODIFY relation-name TO storage-structure ON (key1,key2.....)

The MODIFY command changes the storage structure of a relation from one access form to another. The access methods currently supported by INGRES are discussed in Section 6.5. The indicated keys are domains in relation-name which form the keyed domains. Only the owner of a relation may modify its storage structure.

A secondary Index of a relation can be created by the command

INDEX ON relation-name IS indexname(key1,key2,.....)

It has domains of key1,key2.....,pointer. The domain "pointer" is the unique identifier of a tuple in the indexed relation having the given values of key1, key2..... Consider indexing the AGE domain for the STUDENT relation (APPENDIX 6a). An index named AGEindex for the STUDENT relation might br the binary relation shown in Table 6B.1.

The relation "INDEXNAME" is the treated and accessed just like any other relation, except it is automatically updated when the relation it indexes is updated. Only the owner of a relation may create and destroy indexes for this relation.

AGEINDEX	AGE Pointer
20	Identifier for Bloggs's tuple
24	Identifier for Smith's tuple
20	Identifier for Judy's tuple
23	Identifier for Jones's tuple

Table 6B.1 Binary Relation

6) Miscellaneous

HELP [relation-name of manual - section]

HELP provides information about the system or the database invoked. When called with an option argument which is a command name, HELP returns the appropriate page from the INGRES reference manual, reference [6.13]. If the option argument is a relation name, overall information about that relation together with each attribute, type and length of attribute.

SAVE relation-name UNTIL expiration - date

SAVE is the mechanism by which a user can declare an intention to keep a relation until a specified time.

PURGE database-name

PURGE is a UNIX command which a database administrator may use to delete all relations whose "expiration dates" have passed.

SYSMOD database-name

SYSMOD (modify system relations predetermined storage structures) should

be run on a data base when it is first created and periodically thereafter as relations are created and the data base grows. This will remove most overflow pages and improve system response time. Only the data base administrator has powers to run the SYSMOD command.

Interactive Terminal Monitor Commands

There are a number of commands which may be entered by the user to affect the query buffer or the user's environment. They are all preceded by a backslash ("\"), and all are executed immediately (rather than at execution time like queries). This section gives some of the commonly used commands. A complete list of the monitor commands supported can be found in reference [6.13].

\r (reset) - erases the entire query (rest the query buffer). The previous contents of the buffer are irretrievably lost.

\p (print) - prints the current contents of the buffer.

\e (editor) - causes temporary shift from the monitor to the UNIX text editor (see ED in [6.13]). To return to the INGRES monitor the editor command 'w' followed by 'q' is used.

\g (go) - causes the current contents of the query buffer to be processed. The contents of the query buffer are processed, transmitted to INGRES, and run.

\q (quit) - is used to exit from INGRES.

APPENDIX 8: LISTING OF CURRENT RELATIONS

This Appendix provides a listing of the relations that are currently available in data base "bridge", with their respective attributes, data types and lengths. The attribute "name" in all these relations refers to the name of the bridge. Only the other attributes will be explained in the following section.

Relation: specify
Owner: lfms
Tuple width: 107
Saved until: Tue Apr 22 15:42:19 1986
Number of tuples: 9
Storage structure: paged heap
Relation type: user relation

attribute name	type	length
name	c	20
location	c	20
category	c	10
route	c	30
class	c	2
stat_obligs	c	25

The attributes in relation "specify" are:

- location - the location of the bridge, for example, Slateford;
- category - category of the bridge, for example, road or rail;
- route - the route on which the bridge lies, for example, Edinburgh to Glasgow route.

- class - the categorisation of the bridge according to its importance as outlined in Section 8.1.2.

  stat_obligs - the statutory obligations attached to the bridge.

Relation: arch_dims
Owner: lfms
Tuple width: 40
Saved until: Wed Nov 19 03:16:59 1986
Number of tuples: 9
Storage structure: paged heap
Relation type: user relation

attribute name	type	length
name	c	20
span	f	4
bar_thick	f	4
fill_thick	f	4
half_rise	f	4
qtr_rise	f	4

The attributes in relation "arch_dims" (arch dimensions) are as in Figure 8.1 (Appendix 3) where:

- span - is the span(s) of the bridge, in the case of skew
- spans this is measured parallel to the principal axis of the arch;
- bar_thick - is the thickness of the barrel;
- fill_thick - is the thickness of the arch barrel adjacent to the keystone;

- half_rise _ is the rise of the arch barrel at the crown.
- qtr_rise - is the rise of the arch barrel at the quarter points;

Relation: defects

Owner: lfms

Tuple width: 60

Saved until: Wed Nov 6 16:28:16 1985

Number of tuples: 0

Storage structure: paged heap

Relation type: user relation

attribute name	type	length	keyno.
name	c	12	
component	c	12	
location	c	12	
type	c	12	
orientation	c	12	

The attributes in relation "defects" refer to the following:

- component - the component of the bridge, for example, the barrel;
- location - the location of the defect on the component;
- type - the type of defect, for example, cracking;
- orientation _ the orientation of the defect in the case of cracking.

Relation: vi_inspect

Owner: lfms

Tuple width: 60

Saved until: Tue Apr 22 16:36:48 1986

Number of tuples: 9

Storage structure: paged heap

Relation type: user relation

attribute name	type	length	keyno.
name	c	20	
vi_date	c	10	
vi_details	c	30	

The relation "vi_inspect" has attributes:

- vi_date - date of the visual inspection;
- vi_details - details of the visual inspection.

Relation: maint

Owner: lfms

Tuple width: 60

Saved until: Tue Apr 22 17:11:47 1986

Number of tuples: 9

Storage structure: paged heap

Relation type: user relation

attribute name	type	length
name	c	20
maint_date	c	10

mnt_details c 30

The relation "maint"" has the attributes:

- maint_date - date the maintenance was carried out;
- mnt_details - the details of the maintenance carried out.

Relation: ndt

Owner: lfms

Tuple width: 60

Saved until: Tue Apr 22 16:58:02 1986

Number of tuples: 9

Storage structure: paged heap

Relation type: user relation

attribute name type length

name c 20

ndt_date c 10

ndt_observs c 30

The relation "ndt" (non destructive testing) has the attributes:

- ndt_date - the date the non destructive test was carried out;
- ndt_observs - the non destructive test observations.

```
Relation:           gvw_rest
Owner:              lfms
Tuple width:          27
Saved until:        Mon May 11 03:38:04 1987
Number of tuples:     8
Storage structure:   paged heap
Relation type:       user relation

attribute name    type  length

single              f     4
double              f     4
max_gvw             f     4
veh_type            c     15
```

The relation gvw_rest has attributes:

- single - allowable axle load (tonnes) for single axle load;
- double - allowable axle load (tonnes) for double axle load;
- max_gvw - maximum gross vehicle weight (tonnes) of the C&U vehicles;
- veh_type - type of vehicle.

APPENDIX 10

LISTING OF THE INTERFACE BETWEEN

THE USER AND THE

INGRES RELATIONAL DATA BASE SYSTEM

NOTES

This appendix gives a listing of the interface between INGRES and the user. The statements enclosed between "/*" and "*/" are comment statements as in: /* this is a comment statement */.

```
/*************************************************************
  GLOBAL DECLARATIONS FOR INGRES INTERFACE
*************************************************************/

#include <stdio.h>
#include <ctype.h>
#include "signal.h"
#define READER 0
#define WRITER 1
#define BELL 7
#define CR 10
#define LF 13
#define TRUE 1
#define FALSE 0
#define NL 10
#define SIZE 40
#define PAGESIZE 22
#define EOS '\0'
#define until_quit while(com != 'Q' && com != 'q')
#define STRING1 (sizeof (keytab1)/sizeof (struct key))
#define STRING2 (sizeof (keytab2)/sizeof (struct key))

int to_ingres,from_ingres;
int func(),func1(),domain_c,relation_c;
int flag=0,w_flag=0;
char domain_n[SIZE],ddbase_n[SIZE];
char relation[SIZE];
char s_relation[NL][SIZE],temp[SIZE];
static char c_domain_r[NL] ="name";
```

```
25  static char param[SIZE],dbase_n[SIZE];
26  char range_n[NL] = {'a','b','c','d','e','f','g','h','i','j'};
27  char PATHNAME[NL];
28  FILE  *fpw, *fpr, *fpa;
29  COUNTER = 0;
30  static int  pid,pid1,pid2;

31  static struct key
32  {
33      char   *keyword;
34  }

35  keytab1[] =
36  {
37      "[I]NFORMATION RETRIEVAL",
38      "[D]ECISION SUPPORT SOFTWARE",
39      "[S]HOW RELATIONS IN A DATA BASE",
40      "[L]IST DOMAIN NAMES AND FORMATS",
41      "[P]RINT OUT CONTENTS OF A RELATION",
42      "[V]IEW QUERY BUFFER",
43      "[C]REATING AND MAINTAINING A DATA BASE USING INGRES",
44      "[Q]UIT INGRES"
45  };

46  struct key keytab2[] =
47  {
48      "[C]reate an empty relation",
49      "[F]orming a relation from existing relations",
50      "[D]estroy a relation",
```

```
51      "[E]rase contents of a relation",
52      "[H]ow to copy whole relations to INGRES",
53      "[M]odify system relations",
54      "[T]o destroy a data base",
55      "[S]torage Structures in INGRES",
56      "[Q]uit sub menu to MAIN MENU"
57  };

/****************************
   THE MAIN MODULE
****************************/
58  main()
59  {
60      char com,ch;
61      get_pathname();
62      pid=getpid(); /*get process identity of parent */
63      sprintf(param,"cat /dev/null > hard_copy");
64      system(param);
65      printf("Do you wish to obtain list of available databases? (y/n):");
66      if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y' )
67          {
68              sprintf(param,"cd /usr/ingres/data/base/;ls");
69              system(param);
70          }
71      printf("Enter name of database you wish to consult:\t");
72      scanf("%s",dbase_n);
73      sprintf(param,"creatdb %s",dbase_n);
74      system(param);
```

```
75    sprintf(param,"ingres %s0,dbase_n);
76    if ((pid1 = myopen(param)) == 0)
77       {
78          printf("Can't open ingres\n");
79          exit(1);
80       }
81    if((pid2 = fork())== 0)
82       {
83          close(to_ingres);    /*** READER ***/
84          wait_for_ingres();
85       }
86    if (pid2 == -1)
87       {
88          fprintf(stderr,"Can't fork \n");
89          myclose(pid1);
90          exit(1);
91       }
92    close(from_ingres);  /*** WRITER ***/
93    do
94    {
95       switch(com = menu(keytab1,STRING1,0))
96       {
97          case 'I';  infor_retr();
98          case 'i';  wait_for_signal();
99                     break;
100         case 'D':  decision_support();
101         case 'd':  break;
102         case 'S':  show_rels_in_db();
```

```
103         case 's':   wait_for_signal();
104                     break;
105         case 'L':   list_d();
106         case 'l':   wait_for_signal();
107                     break;
108         case 'P':   print_rel();
109         case 'p':   wait_for_signal();
110                     break;
111         case 'V':   view_buf();
112         case 'v':   wait_for_signal();
113                     break;
114         case 'C':   create_maint();
115         case 'c':   break;
116         case 'Q':
117         case 'q':
118         case 'Q':   clean_up_ingres();
119         case 'q':   wait_for_signal();
120                     stop_ingres();
121                     myclose(pid1);
122                     break;
123         default:    printf("No such facility, Please try again.\n");
124                     break;
125       }
126     }
127     until_quit;
128     fprintf(stderr,"\nCHEERIO : ");  /* wrap up */
129     system("echo $HOME");
130     system("date");
```

```
131  }

/********************
 *WAIT_FOR_INGRES : an '*' => ingres is ready to input the next line.
********************/
132  wait_for_ingres()
133  {
134     int lcount;
135     char c;
136     lcount = 0;
137     if( ( fpw = fopen("ingres_junk","w")) == NULL)
138        {
139           fprintf(stderr,"\n\tError: Can't open ingres_junk(w)\n");
140           exit(1);
141        }
142     do
143     {
144        while( read(from_ingres,&c,1) != 1);
145        if ( c == BELL )
146        {
147           while ( c != '*')
148           {
149              read(from_ingres,&c,1);
150              putchar(c);
151              write(fileno(fpw),&c,1);
152        }
153        lcount = 0;
154        kill(pid,SIGINT); /*send synchronisation signal to parent */
```

```
155         continue;
156       }
157     putchar(c);
158     write(fileno(fpw),&c,1);
159     if( c == '\n') lcount++;
160     if( lcount == PAGESIZE )
161       {
162         kill(pid2,SIGURG); /* send pagesize signal */
163         lcount = 0;
164         continue;
165       }
166     }
167     while(TRUE);
168     fflush(fpw);
  169     fclose(fpw);
  170   }

/*********************
 * WAIT_FOR_SIGNAL : synchronisation routine
*********************/
  171   wait_for_signal()
  172   {
  173     do
  174     {
  175       signal(SIGINT, func);   /* catch synchronisation signal */
  176       signal(SIGURG, func1);  /* catch pagesize signal */
  177     }           /*and act accordingly*/
  178     while ( flag == 0 );
```

```
179     flag=0;
180  }

181  func()
182  {
183     flag=1;
184  }

185  /* Gives a screenfull at a time */
186  func1()
187  {
188     char ch,s[4];
189     kill(pid2,SIGSTOP);
190     printf("\n[Enter 'm' for MORE]\n");
191     do
192     {
193        ch = wait_for_more();
194     }
195     while (ch != 'm' && ch != 'M' );
196     kill(pid2,SIGCONT);
197  }


/*******
 *MENU : list of existing utilities.
*******/
   198  menu(p_menu,limit,m_flag) struct key *p_menu; int limit;
   199  {
   200     char ch,s[4];
   201     int i;
```

```
202     char c;
203     if ( w_flag == 0 ) wait_for_signal();
204     if ( w_flag == 1 )
205     {
206        printf("\nDo you wish to obtain a copy of your request? (y/n):\t");
207        if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y' )
208        {
209           if( (fpr = fopen("ingres_junk","r")) == NULL)
210           {
211              fprintf(stderr,"\nError: Can't open ingres_junk(r)\n)");
212              exit(1);
213           }
214           if( (fpa = fopen("hard_copy","a")) == NULL)
215           {
216              fprintf(stderr,"\nError: Can't open hard_copy(a)\n");
217              exit(1);
218           }
219           while ((c = getc(fpr)) != EOF) putc(c,fpa);
220           sprintf(param,"cat /dev/null > ingres_junk");
221           system(param);
222           fflush(fpr);
223           fflush(fpa);
224           fclose(fpr);
225           fclose(fpa);
226        }
227        if ( ch == 'N' || ch == 'n')
228        {
229           sprintf(param,"cat /dev/null > ingres_junk");
```

```
230         system(param);
231       }
232       if ( m_flag == 0 )printf("\n\t[Enter 'm' for MAIN MENU]\t");
233       else printf("\n\t[Enter 'm' for SUB_MENU]\t");
234       do
235       {
236         ch = wait_for_more();
237       }
238       while ( ch != 'm' && ch != 'M' );
239     }
240     w_flag = 1 ;
241     printf("\n\n");
242     for(i=1;i<=limit;i++) printf("\t%s\n",*p_menu++);
243     printf("\nEnter square bracketed letter for your selection.\n");
244     printf("\nSelect: ");
245     return(getcom());
246   }

/* GET SELECTION FROM MENU */
247   getcom()  /* get selection from menu */
248   {
249     char   c;
250     while((c = getchar()) == NL);
251     printf("\n");
252     return(c);
253   }
```

```
/* RUN INGRES AND THEN CLEAR WORSPACE : */
  254  run_ingres()
  255  {
  256      write(to_ingres,"\\g\\r\n",5);
  257  }

/*  DESTROY TEMPORARY RELATIONS */
  258  clean_up_ingres()
  259  {
  260      int i;
  261      for( i = 0; i < COUNTER; i++)
  262      {
  263          sprintf(param,"destroy requested%d\n",i);
  264          write(to_ingres,param,strlen(param) );
  265      }
  266      run_ingres();
  267  }

/*  STOP_INGRES : terminates ingres from the background.  */
  268  stop_ingres()
  269  {
  270      write(to_ingres,"\\q\n",3);
  271  }

/*  MYOPEN : form 2 pipes connecting parent and ingres.  */
  272  myopen(cmd)
  273  char *cmd;
  274  {
  275      int p1[2],p2[2],pid1;
```

```
276     if(pipe(p1)<0 ||pipe(p2)<0)return(0);
277     if((pid1 = fork()) == 0)
278     {
279        close(p1[WRITER]);
280        close(p2[READER]);
281        if(p2[WRITER]!=1)
282        {
283           dup2(p2[WRITER],1);
284           close(p2[WRITER]);
285        }
286        if(p1[READER]!=0)
287        {  dup2(p1[READER],0);
288           close(p1[READER]);
289        }
290        execl("/bin/sh","sh","-c",cmd,0);
291        exit(1);
292     }
293     if(pid1 == -1)
294     {
295        close(p1[READER]);
296        close(p1[WRITER]);
297        close(p2[READER]);
298        close(p2[WRITER]);
299        return(0);
300     }
301     close(p2[WRITER]);
302     close(p1[READER]);
303     to_ingres = p1[WRITER];
```

```
304     from_ingres=p2[READER];
305     return(pid1);
306  }

/*  MYCLOSE : closes program opened by myopen  */
 307  myclose(pid1)
308  int pid1;
309  {
310     register r;
311     int status;
312     close(to_ingres);  /*close file descripters */
313     close(from_ingres);
314     signal(SIGINT,SIG_IGN);
315     signal(SIGQUIT,SIG_IGN);
316     signal(SIGHUP,SIG_IGN);
317     kill(pid2,SIGKILL); /* terminate LISTENER */
318     while((r = wait(&status))!= pid1 && r!= -1);
319     if( r == -1) status = -1;
320     return(status);
321  }

322  get_number()  /* get numerical selection from user */
323  {
324     char num;
325     while(TRUE)
326     {
327        while ( ( (num = getchar() ) == LF) || (num == CR) );
328        switch (num)
```

```
329     {
330         case '1':   return(num);
331         case '2':   return(num);
332         case '3':   return(num);
333         case '4':   return(num);
334         case '5':   return(num);
335         case '6':   return(num);
336         case '7':   return(num);
337         case '8':   return(num);
338         case '9':   return(num);
339       default:    printf("Please enter digit\n");
340       }
341   }
342 }
343 char *get_selection() /* return selection */
344 {
345   static char ch[SIZE];
346   char i,c;
347   do
348     c = getchar();
349   while (  !isalnum( c ));
350   for ( i = 0; (( c != LF) && ( c != CR ) ); )
351   {
352     if (isdigit(c) ) ch[i++] = c;
353     c = getchar();
354   }
355   ch[i] = '\0';
```

```
356     return ( ch );
357  }

/*****************************************************
   DECLARE DATA BASE BRIDGE'S DETAILS
*****************************************************/

358  char *rel_name(n) int n; /* return name of n-th relation */
359  {
360     static char *name[] =
361     {
362     "specify","ndt","maint","vi_Inspect"
363     };
364     return(name[n]);
365  }
366

367     /* return n-th name of specify domains */
368  char *specify_dom_name(n) int n;
369  {
370     static char *specify_domains[] =
371     {
372     "name","location","category","route"
373     };
374     return(specify_domains[n]);
375  }

376  char *ndt_dom_name(n) int n;
377     /* return n-th name of ndt domains */
```

```
{
    static char *ndt_domains[] =
    {
    "name","ndt_date","ndt_observs"
    };
    return(ndt_domains[n]);
}

char *maint_dom_name(n) int n;
    /* return n-th name of maint domains */
{
    static char *maint_domains[] =
    {
    "name","maint_date","mnt_details"
    };
    return(maint_domains[n]);
}

char *vi_inspect_dom_name(n) int n;
    /* return n-th name of vi_inspect domains */
{
    static char *vi_inspect_domains[] =
    {
    "name","vi_date","vi_details"
    };
    return(vi_inspect_domains[n]);
}
```

```
/*************************************************************
 DECLARE VARIABLES FOR VARIOUS SELECTIONS
*************************************************************/
  404  char temp_memory_selection[SIZE];
  405  char memory_selection[NL][SIZE];
  406  char rel_selection[NL];
  407  char specify_selection[NL];
  408  char ndt_selection[NL];
  409  char maint_selection[NL];
  410  char vi_inspect_selection[NL];
  411  char qual_selection;
  412  char pi,si, ch;
  413  int rel_c, i, ti,tj,fi;
  414  int check_flag = 0;
  415  COUNT = 0;

/************************************
 INFORMATION RETRIEVAL
************************************/
  416  infor_retr()
  417  {
  418      char *mesg4 = "\n\
  419      Which of the following do you require to consult?\n\
  420      1.Specification   2. Ndt   3.Maintenance   4. Visual Inspection\n\
  421      5. None of the above\n";
  422      printf( mesg4);
  423      printf("\nSelect by Number:   ");
  424      strcpy( rel_selection, get_selection() );
```

```
425     rel_c = strlen(rel_selection);
426     for ( i = 0; i < rel_c;)
427     {
428        ti = rel_selection[i++] - '0';
429        range(range_n[ti-1],rel_name(ti-1));
430     }
431     sprintf(param,"retrieve into requested%d(\n",COUNTER);
432     write(to_ingres,param,strlen(param));
433     for ( i = 0; i < rel_c; )
434     {
435        fi = i;
436        si = rel_selection[i++];
437        temp_memory_selection[COUNT++] = si;
438        ti =(si - '0') -1 ;
439        switch ( si)
440        {
441           case '1':   specify(range_n[ti],(rel_c-1),fi);
442                       break;
443           case '2':   ndt(range_n[ti],(rel_c-1),fi);
444                       break;
445           case '3':   maint(range_n[ti],(rel_c-1),fi);
446                       break;
447           case '4':   vi_inspect(range_n[ti],(rel_c-1),fi);
448                       break;
449           default:    break;
450        }
451     }
452     write(to_ingres,")\n",2);
```

```
453     printf("\nDo you wish to qualify your requests?  (y/n):\t");
454     if ( ( ch = yes_or_no() ) == 'N' || ch == 'n' )
455     {
456        if ( rel_c > 1 )
457        {
458        sprintf(param,"where\n");
459        write(to_ingres,param,strlen(param) );
460        for ( i = 0; i < rel_c-1; i++ )
461        {
462           pi = rel_selection[i];
463           si = rel_selection[(i+1)];
464           fi = i;
465           common_domain(fi,pi,si,0);
466        }
467        }
468     temp_memory_selection[COUNT++] = '\0';
469     if (COUNTER != 0) query_memory(0);
470     print_requested_relation(0);
471     COUNT = 0;
472     }
473     else with_qual();
474  }

475  specify(s1,s2,s3)
476  char *s1;
477  int  *s2, *s3;
478  {
479     char *mesg5 = "\n\
```

```
480     Of which of these on Specification do you require information?\n\
481i    1. Name       2. Location   3.Category    4. Route\n";
482     int j, pj,tj;
483     printf(mesg5);
484     printf("\nSelect by Number:   ");
485     strcpy(specify_selection,get_selection() );
486     domain_c = strlen(specify_selection);
487     for ( j = 0; j < domain_c; j++)
488     {
489       pj = specify_selection[j];
490       tj = pj - '0';
491       temp_memory_selection[COUNT++] = pj;
492       sprintf(param,"%c.%s",s1,specify_dom_name(tj-1) );
493       write(to_ingres,param,strlen(param));
494       if ( s3 == s2 && j == (domain_c -1) )
495          write(to_ingres,"\n",1);
496       else write(to_ingres,",\n",2);
497     }
498  }

499  ndt(s1,s2,s3)
500  char *s1;
501  int  *s2, *s3;
502  {
503     char *mesg6 = "\n\
504     Of which of these on non-destructive testing (Ndt) do you wish\n\
505     to obtain information?\n\
506     1. Name       2. Ndt Date      3.Ndt Observation\n";
```

```
507     int j,pj, tj;
508     printf(mesg6);
509     printf("\nSelect by Number:   ");
510     strcpy(ndt_selection,get_selection() );
511     domain_c = strlen(ndt_selection);
512     for ( j = 0; j < domain_c; j++)
513     {
514        pj = ndt_selection[j];
515        tj = pj - '0';
516        temp_memory_selection[COUNT++] = pj;
517        sprintf(param,"%c.%s",s1,ndt_dom_name(tj-1) );
518        write(to_ingres,param,strlen(param));
519        if ( s3 == s2 && j == (domain_c -1) )
520           write(to_ingres,"\n",1);
521           else write(to_ingres,",\n",2);
522     }
523  }

524  maint(s1,s2,s3)
525  char *s1;
526  int *s2, *s3;
527  {
528     char *mesg7 = "\n\
529     Of which of the following on Maintenance do you wish to obtain\n\
530     information?\n\
531     1.Name      2. Maintenace Date   3. Maintenance Details\n";
532     int j, pj, tj;
533     printf(mesg7);
```

```
534     printf("\nSelect by Number:   ");
535     strcpy(maint_selection,get_selection() );
536     domain_c = strlen( maint_selection);
537     for ( j = 0; j < domain_c; j++)
538     {
539        pj = maint_selection[j];
540        tj = pj - '0';
541        temp_memory_selection[COUNT++] = pj;
542        sprintf(param,"%c.%s",s1,maint_dom_name(tj-1) );
543        write(to_ingres,param,strlen(param));
544        if ( s3 == s2 && j == (domain_c -1) )
545           write(to_ingres,"\n",1);
546        else write(to_ingres,",\n",2);
547     }
548  }

549  vi_inspect(s1,s2,s3)
550  char *s1;
551  int *s2, *s3;
552  {
553     char *mesg8 = "\n\
554     Of which of the following on Visual Inspection do you wish to\n\
555     obtain information?\n\
556     1. Name      2. Visual Inspection Date    3. Details\n";
557     int j, pj, tj;
558     printf(mesg8);
559     printf("\nSelect by Number:   ");
560     strcpy(vi_inspect_selection,get_selection() );
```

```
561     domain_c = strlen(vi_inspect_selection);
562     for ( j = 0; j < domain_c; j++)
563     {
564        pj = vi_inspect_selection[j];
565        tj = pj - '0';
566        temp_memory_selection[COUNT++] = pj;
567        sprintf(param,"%c.%s",s1,vi_inspect_dom_name(tj-1));
568        write(to_ingres,param,strlen(param));
569        if ( s3 == s2 && j == (domain_c -1) )
570           write(to_ingres,"\n",1);
571        else write(to_ingres,",\n",2);
572     }
573  }
```

```
/************************************
  EQUALITY OPERATORS
************************************/
```

```
574  char *equality(n) int n;
575  {
576     static char *equality[] =
577     {
578        "=","!=",">",">=","<","<="
579     };
580     return(equality[n]);
581  }
```

```
582  char *equality_mesg = "\n\
583  Which one of these qualifications do you wish to apply?\n\
```

```
584  1. Equal                4. Greater than or equal\n\
585  2. Not Equal            5. Less than\n\
586  3. Greater than         6. Less than or equal\n";

/************************************
  LOGICAL OPERATORS
************************************/
587  char *logical(n) int n;
588  {
589      static char *logical[] =
590      {
591          "and","or","not"
592      };
593      return(logical[n]);
594  }

595  char *logical_mesg= "\n\
596  Which one of these logical operators do you wish to apply?\n\
597  1. And        2. Or        3. Not\n";

598  char *qual_count_mesg = "\n\
599  How many qualifications do you wish to place on this domain?\t";

600  char *qual_or_not_mesg = "\n\
601  Do you wish to qualify %s (y/n)?\t";

602  char *qual_mesg = "\n\
603  Please enter your qualification.\t";

604  char *help_qual_mesg = "\n\
```

```
605  Which one of the following pattern matching constructs\n\
606  do you wish to use for your qualification?\n\
607  1. Know the full qualification\n\
608  2. Know the first characters of the qualification\n\
609  3. Know the middle part of the qualification\n\
610  4. Know only the last characters of the qualification\n";
611  int ne,nl;

612  int qual_flag = 0;
613  int qual_check_flag = 0;
614  int logical_selection;
615  char equality_selection;
616  char qualification[SIZE];
617  char temp_qual_memory[SIZE];
618  char qual_memory[NL][SIZE];
619  char qual_count;

/*************************************************************
 QUALIFICATIONS FOR INFORMATION RETRIEVAL
*************************************************************/

620  with_qual()
621  {
622      char si;
623      int rel_c, i, ti,fi;
624      rel_c = strlen(rel_selection);
625      for ( i = 0; i < rel_c; )
626      {
627          fi = i;
628          si = rel_selection[i++];
```

```
629     ti = (si -'0') -1 ;
630     switch ( si )
631     {
632        case '1':   specify_qual(range_n[ti]);
633                    break;
634        case '2':   ndt_qual(range_n[ti]);
635                    break;
636        case '3':   maint_qual(range_n[ti]);
637                    break;
638        case '4':   vi_inspect_qual(range_n[ti]);
639                    break;
640        default:    break;
641     }
642  }
643  if ( rel_c > 1)
644  {
645     for ( i = 0; i < rel_c -1; i++)
646     {
647        pi = rel_selection[i];
648        si = rel_selection[i+1];
649        fi = i;
650        common_domain(fi,pi,si,1);
651     }
652  }
653  temp_memory_selection[COUNT++] = '\0';
654  if (COUNTER != 0) query_memory(1);
655  print_requested_relation(1);
656  COUNT = 0;
```

```
657     temp_qual_memory[0] = '\0';
658     qual_flag = 0;
659     qual_check_flag = 0;
660  }

661  specify_qual(s1)
662  char *s1;
663  {
664     char ch;
665     int j,pj ,tj ,i;
666     printf("\nDo you wish to qualify specifications?  (y/n):\t");
667     if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y' )
668     {
669        temp_memory_selection[COUNT++] = 'Y';
670        domain_c = strlen(specify_selection);
671        for ( j = 0; j < domain_c; j++)
672        {
673           pj = specify_selection[j];
674           tj = pj- '0' -1;
675           printf(qual_or_not_mesg,specify_dom_name(tj) );
676           if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y' )
677           {
678           temp_memory_selection[COUNT++] = 'Y';
679           if (qual_check_flag == 0 )
680           {
681              sprintf(param,"where\n");
682              write(to_ingres,param,strlen(param));
683              qual_check_flag = 1;
```

```
684         }
685         printf(qual_count_mesg);
686         qual_count = get_number() ;
687         temp_memory_selection[COUNT++] = qual_count;
688         for ( i = 1; i <= (qual_count - '0'); i++ )
689         {
690           if( (qual_flag == 0) )
691           {
692             equality_operator( s1,specify_dom_name(tj) );
693             qualify();
694             qual_flag = 1;
695           }
696           else
697           {
698             logical_operator();
699             equality_operator(s1,specify_dom_name(tj) );
700             qualify();
701           }
702         }
703       }
704       else temp_memory_selection[COUNT++] = 'N';
705     }
706     }
707     else temp_memory_selection[COUNT++] = 'N';
708 }

709 ndt_qual(s1)
710 char *s1;
```

```
711 {
712     char ch;
713     int j,pj ,tj ,i;
714     printf("\nDo you wish to qualify non destructive tests? (y/n):\t");
715     if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y' )
716     {
717         temp_memory_selection[COUNT++] = 'Y';
718         domain_c = strlen(ndt_selection);
719         for ( j = 0; j < domain_c; j++)
720         {
721             pj = ndt_selection[j];
722             tj = pj- '0' -1;
723             printf(qual_or_not_mesg,ndt_dom_name(tj) );
724         if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y' )
725         {
726             temp_memory_selection[COUNT++] = 'Y';
727             if (qual_check_flag == 0 )
728             {
729                 sprintf(param,"where\n");
730                 write(to_ingres,param,strlen(param));
731                 qual_check_flag = 1;
732             }
733         printf(qual_count_mesg);
734         qual_count = get_number() ;
735         temp_memory_selection[COUNT++] = qual_count;
736         for ( i = 1; i <= (qual_count -'0'); i++ )
737         {
738             if( (qual_flag == 0) )
```

```
739             {
740                equality_operator( s1,ndt_dom_name(tj) );
741                qualify();
742                qual_flag = 1;
743             }
744             else
745             {
746                logical_operator();
747                equality_operator(s1,ndt_dom_name(tj) );
748                qualify();
749             }
750          }
751       }
752          else temp_memory_selection[COUNT++] = 'N';
753       }
754    }
755    else temp_memory_selection[COUNT++] = 'N';
756 }

757 maint_qual(s1)
758 char *s1;
759 {
760    char ch;
761    int j,pj ,tj ,i;
762    printf("\nDo you wish to qualify maintenance? (y/n):\t");
763    if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y' )
764    {
765       temp_memory_selection[COUNT++] = 'Y';
```

```
766        domain_c = strlen(maint_selection);
767        for ( j = 0; j < domain_c; j++)
768        {
769           pj = maint_selection[j];
770           tj = pj- '0' -1;
771        printf(qual_or_not_mesg,maint_dom_name(tj) );
772        if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y' )
773        {
774           temp_memory_selection[COUNT++] = 'Y';
775           if (qual_check_flag == 0 )
776           {
777              sprintf(param,"where\n");
778              write(to_ingres,param,strlen(param));
779              qual_check_flag = 1;
780           }
781        printf(qual_count_mesg);
782        qual_count = get_number() ;
783        temp_memory_selection[COUNT++] = qual_count;
784        for ( i = 1; i <= (qual_count -'0'); i++ )
785        {
786           if( (qual_flag == 0) )
787           {
788              equality_operator( s1,maint_dom_name(tj) );
789              qualify();
790              qual_flag = 1;
791           }
792           else
793           {
```

```
794                 logical_operator();
795                 equality_operator(s1,maint_dom_name(tj) );
796                 qualify();
797               }
798             }
799               }
800               else temp_memory_selection[COUNT++] = 'N';
801             }
802           }
803           else temp_memory_selection[COUNT++] = 'N';
804 }

805 vi_inspect_qual(s1)
806 char *s1;
807 {
808     char ch;
809     int j,pj ,tj ,i;
810     printf("\nDo you wish to qualify visual inspection? (y/n):\t");
811     if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y' )
812     {
813         temp_memory_selection[COUNT++] = 'Y';
814         domain_c = strlen(vi_inspect_selection);
815         for ( j = 0; j < domain_c; j++)
816         {
817             pj = vi_inspect_selection[j];
818             tj = pj- '0' -1;
819             printf(qual_or_not_mesg,vi_inspect_dom_name(tj) );
820             if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y' )
```

```
821         {
822             temp_memory_selection[COUNT++] = 'Y';
823             if (qual_check_flag == 0 )
824             {
825                 sprintf(param,"where\n");
826                 write(to_ingres,param,strlen(param));
827                 qual_check_flag = 1;
828             }
829             printf(qual_count_mesg);
830             qual_count = get_number() ;
831             temp_memory_selection[COUNT++] = qual_count;
832             for ( i = 1; i <= (qual_count -'0'); i++ )
833             {
834                 if( (qual_flag == 0) )
835                 {
836                     equality_operator( s1,vi_inspect_dom_name(tj) );
837                     qualify();
838                     qual_flag = 1;
839                 }
840                 else
841                 {
842                     logical_operator();
843                     equality_operator(s1,vi_inspect_dom_name(tj) );
844                     qualify();
845                 }
846             }
847         }
848         else temp_memory_selection[COUNT++] = 'N';
```

```
849         }
850     }
851     else temp_memory_selection[COUNT++] = 'N';
852 }

/*********
* EQUALITY: This routine adds the equality operator.
**********/

853 equality_operator(s1,s2)
854 char *s1 ;
855 char *s2;
856 {
857     printf(equality_mesg);
858     printf("\n\tSelect by Number:       ");
859     equality_selection = get_number() ;
860     ne = ( (equality_selection -'0') -1 );
861     temp_memory_selection[COUNT++] = equality_selection;
862     sprintf(param,"%c.%s %s ",s1,s2,equality(ne) );
863     write(to_ingres,param,strlen(param) );
864 }

/************
LOGICAL : This routine adds the logical operator.
************/

865 logical_operator()
866 {
867     printf(logical_mesg);
868     printf("\n\tSelect by Number:       ");
869     logical_selection = get_number() ;
```

```
870     nl = ( (logical_selection -'0') -1 );
871     temp_memory_selection[COUNT++] = logical_selection;
872     sprintf(param,"%s ",logical(nl) );
873     write(to_ingres,"\n",1);
874     write(to_ingres,param,strlen(param) );
875  }

/************
 QUALIFY: This routine adds the qualification
************/
876  qualify()
877  {
878     char qi;
879     int qual_type_count;
880     printf(help_qual_mesg);
881     qual_selection = get_number() ;
882     temp_memory_selection[COUNT++] = qual_selection;
883     printf(qual_mesg);
884     scanf("%s",qualification);
885     strcat(temp_qual_memory,qualification);
886     switch(qual_selection)
887     {
888        case '1':   sprintf(param,"\"%s\"\n",qualification);
889                    break;
890        case '2':   sprintf(param,"\"%s\*\"\n",qualification);
891                    break;
892        case '3':   sprintf(param,"\"\*%s\*\"\n",qualification);
893                    break;
```

```
894       case '4':   sprintf(param,"\"\*%s\"\n",qualification);
895                   break;
896       default:    break;
897    }
898    write(to_ingres,param,strlen(param) );
899 }

900 common_domain(s1,s2,s3,c_dom_flag)
901 char *s1, s2,s3;
902 {
903    int ti, tj;
904    ti = s2 - '0' -1;
905    tj = s3 -'0' -1;
906    if ( (*s1 == 0 ) && (c_dom_flag == 0 ) )
907    {
908       sprintf(param,"%c.%s= %c.%s\n",range_n[ti],
909       c_domain_r,range_n[tj],c_domain_r);
910       write(to_ingres,param,strlen(param) );
911    }
912    else
913    {
914       sprintf(param,"and %c.%s= %c.%s\n",range_n[ti],
915       c_domain_r,range_n[tj],c_domain_r);
916       write(to_ingres,param,strlen(param) );
917    }
918 }

919 query_memory(qual_marker)
920 {
```

```
921     int i;
922     if (qual_marker == 1)
923     {
924     for ( i = 0; i < COUNTER; i++)
925        {
926        if( (strcmp(memory_selection[i],temp_memory_selection) == 0)
927           && (strcmp(qual_memory[i],temp_qual_memory) == 0) )
928           {
929              check_flag = 1;
930              write(to_ingres,"\\r\n",3);
931              wait_for_signal();
932              sprintf(param,"print requested%d",i);
933              write(to_ingres,param,strlen(param) );
934           }
935        }
936     }
937     else for ( i = 0; i < COUNTER; i++)
938        {
939           if (strcmp(memory_selection[i],temp_memory_selection) == 0 )
940              {
941                 check_flag = 1;
942                 write(to_ingres,"\\r\n",3);
943                 wait_for_signal();
944                 sprintf(param,"print requested%d",i);
945                 write(to_ingres,param,strlen(param) );
946              }
947        }
948  }
```

```
print_requested_relation(qual_mark)
{
    if (check_flag == 0)
        {
            sprintf(param,"print requested%d\n",COUNTER );
        write(to_ingres,param,strlen(param) );
        if (qual_mark == 1)
          {
            strcpy(memory_selection[COUNTER],temp_memory_sclection);
            strcpy(qual_memory[COUNTER],temp_qual_memory);
          }
        else strcpy(memory_selection[COUNTER],temp_memory_selection);
        COUNTER++;
    }
    else check_flag = 0;
    run_ingres();
}

char *bar_factor_mesg = "\n\
1.  Granite and Whitstone whether random or coursed and all\n\
    built-in-course masonry except limestone, all with large\n\
    shaped voussoirs.\n\
2.  Concrete or engineering bricks and similar sized masonry\n\
    (not limestone).\n\
3.  Limestone, whether random or coursed, good random masonry\n\
    and building bricks, all in good conditon.\n\
4.  Masonry of any kind in poor conditon (many voussoirs flaking\n\
    or badly spalling, shearing etc.). Some discretion is permitted\n\
```

```
976       if the dilapidation is only moderate.\n";

977  char *fill_factor_mesg = "\n\
978  1.  Concrete\n\
979  2.  Grouted materials (other than those with a clay content)\n\
980  3.  Well compacted materials\n\
981  4.  Weak materilas evidenced by tracking of the carriageway surface.\n\
982  5.  When assessing an arch for construction and use vehicles, and\n\
983       details of the fill are unknown or there is evidence of weakness\n\
984       from the condition of the road surface\n";

985  char *width_factor_mesg = "\n\
986  1.  Joints with widths up to 6mm\n\
987  2.  Joints with widths between 6mm and 12.5mm\n\
988  3.  Joints with widths over 12.5mm\n";

989  char *mort_factor_mesg = "\n\
990  1.  Mortar in good condition\n\
991  2.  Loose or friable mortar\n";

992  char * depth_factor_mesg = "\n\
993  1.  Unpointed joints, pointing in poor condition and joints with up\n\
994       to 12.5mm from the edge insufficiently filled.\n\
995  2.  Joints with from 12.5mm to one tenth of the thickness of the barrel\n\
996       insufficiently filled.\n\
997  3.  Joints insufficiently filled for more than one-tenth the thickness\n\
998       of the barrel.\n\
999  4.  Pointed joints in good condition\n";

1000  char * mort_thick_mesg = "\n\
```

```
1001  Enter estimate of thickness of missing motar(mm) (e.g. 12.5)\n";

1002  char * alarm_mesg = "\n\
1003  Immediate consideration should be given to the repair or\n\
1004  reconstruction of the bridge\n";
1005  int choice; /* choice of factors */

1006  char *dims_dom_name(n) int n; /*n#*/
1007  {
1008      static char *dims_domains[] =
1009      {
1010          "name","span","bar_thick","fill_thick","half_rise","qtr_rise"
1011      };
1012      return(dims_domains[n]);
1013  }

1014  decision_support()
1015  {
1016      FILE *fd;
1017      char DIMS[SIZE];
1018      char name[NL], ch;
1019      float span; bar_thick,fill_thick,half_rise,qtr_rise;
1020      float mort_thick,MoF, DF,BF,FF,WF,MF,JF,CF;
1021      float PAL,MAL,sum,bnum,bdenom,quot;
1022      float SRR,SRF,PF,RR;
1023      char bridge_name[SIZE];
1024      char bridge_location[SIZE];
1025      printf("\nPlease enter name of bridge you wish to assess:   ");
1026      scanf("%s",bridge_name);
```

```
1027    printf("\nEnter location of bridge:        ");
1028    scanf("%s",bridge_location);
1029    rel_c = 8;
1030    sprintf(param,"range of %c is arch_dims\n",range_n[rel_c]);
1031    write(to_ingres,param,strlen(param) );
1032    sprintf(param,"range of %c is specify\n",range_n[rel_c+1] );
1033    write(to_ingres,param,strlen(param) );
1034    sprintf(param,"retrieve into temp_dims(\n");
1035    write(to_ingres,param,strlen(param) );
1036    for ( i = 1; i <= 5; i++)
1037      {
1038        sprintf(param,"%c.%s",range_n[rel_c],dims_dom_name(i) );
1039        write(to_ingres,param,strlen(param) );
1040        if ( i != 5 ) write(to_ingres,",\n",2);
1041          else write(to_ingres,"\n",1);
1042      }
1043    write(to_ingres,")\n",2);
1044    sprintf(param,"where %c.name = \"\*%s\*\"\n",
1045    range_n[rel_c],bridge_name );
1046    write(to_ingres,param,strlen(param) );
1047    sprintf(param,"and %c.location = \"\*%s\*\"\n",
1048    range_n[rel_c+1], bridge_location);
1049    write(to_ingres,param,strlen(param) );
1050    run_ingres();
1051    wait_for_signal();
1052    sprintf(param,"copy temp_dims (\n");
1053    write(to_ingres,param,strlen(param) );
1054    for (i = 1; i <= 5; i++)
```

```
1055        {
1056            sprintf(param,"%s = c10",dims_dom_name(i) );
1057            write(to_ingres,param,strlen(param) );
1058            if ( i != 5 ) write(to_ingres,",\n",2);
1059            else write(to_ingres,"\n",1);
1060        }
1061        write(to_ingres,")\n",2);
1062        sprintf(param,"into \"%s/temp_file\"",PATHNAME);
1063        write(to_ingres,param,strlen(param) );
1064        sprintf(param,"destroy temp_dims");
1065        write(to_ingres,param,strlen(param) );
1066        run_ingres();
1067        wait_for_signal();
1068        if ( ( fd = fopen("temp_file", "r")) == NULL)
1069            {
1070                fprintf(stderr,"\nError: Can't open temp_file(r)\n");
1071                exit(1);
1072            }
1073        for(i = 0; i < SIZE;)
1074            {
1075                if (fscanf(fd,"%c",&DIMS[i++]) <= 0)
1076                    {
1077                        DIMS[i] = EOS;
1078                        break;
1079                    }
1080            }
1081        sscanf(DIMS,"%10f %10f %10f %10f %10f",
1082            &span,&bar_thick,&fill_thick,&half_rise,&qtr_rise);
```

```
1083    if (span == 0 && bar_thick == 0 && fill_thick == 0 &&
1084       half_rise == 0 && qtr_rise == 0)
1085       {
1086          printf("Sorry, no dimensions available for this bridge");
1087          exit(1);
1088       }
1089    printf("Thickness of barrel is %10.2fmetres\n",bar_thick);
1090    printf(depth_factor_mesg);
1091    printf("\nSelect by number the construction of joint:   ");
1092    scanf("%d",&choice);
1093    if (choice == 1) DF = 0.9;  /* DF - Depth Factor */
1094    if (choice == 2) DF = 0.8;
1095    if (choice == 3)
1096       {
1097          printf(mort_thick_mesg);
1098          scanf("%5f",&mort_thick);
1099          bar_thick = bar_thick - (mort_thick/100.0);
1100       }
1101    if (choice == 4) DF = 1.0;
1102    sum = bar_thick + fill_thick;
1103    bnum = 740.0 * (pow(sum,2.0) );
1104    bdenom = pow (span,1.3);
1105    PAL = bnum / bdenom ; /* Provisonal axle loading */
       /* The value of SRR is calculated from Figure 3.3, see Appendix 3) */
1106    SRR = span / half_rise; /* Span/Rise Ratio */
1107    if ( SRR <= 4.0 ) SRF = 1.0;  /* SRF - Span/Rise Factor */
1108    if ( SRR > 4.0 && SRR <= 5.0 ) SRF = 0.855 + 0.145*(5.0 - SRR);
1109    if ( SRR > 5.0 && SRR <= 6.0 ) SRF = 0.750 + 0.105*(6.0 - SRR);
```

```
1110    if ( SRR > 6.0 && SRR <= 7.0 ) SRF = 0.675 + 0.075*(7.0 - SRR);
1111    if ( SRR > 7.0 && SRR <= 8.0 ) SRF = 0.615 + 0.060*(8.0 - SRR);
1112    RR = half_rise / qtr_rise;
1113    if ( RR <= 0.75 ) PF = 1.0; /* PF - Profile Factor */
1114    else
1115    PF = pow(((half_rise-qtr_rise)/half_rise),0.6) * 2.3;
1116    printf(bar_factor_mesg);
1117    printf("\nSelect by number the arch barrel details:   ");
1118    scanf("%d",&choice);
1119    if (choice == 1) BF = 1.5;  /* BF - Barrel Factor */
1120    if (choice == 2) BF = 1.2;
1121    if (choice == 3) BF = 1.0;
1122    if (choice == 4) BF = 0.7;
1123    printf(fill_factor_mesg);
1124    printf("\nSelect by number the filling:   ");
1125    scanf("%d",&choice);
1126    if (choice == 1) FF = 1.0;  /* FF - Fill Factor */
1127    if (choice == 2) FF = 0.9;
1128    if (choice == 3 || choice == 5) FF = 0.7;
1129    if (choice == 4) FF = 0.5;
1130    MF = ((BF * bar_thick)+(FF * fill_thick))/(bar_thick+fill_thick);
1131    printf(width_factor_mesg);
1132    printf("\nSelect by number the width of joints:   ");
1133    scanf("%d",&choice);
1134    if (choice == 1) WF = 1.0;  /* WF - Width Factor */
1135    if (choice == 2) WF = 0.9;
1136    if (choice == 3) WF = 0.8;
1137    printf(mort_factor_mesg);
```

```
1138    printf("\nSelect by number the condition of joint:   ");
1139    scanf("%d",&choice);
1140    if (choice == 1) MoF = 1.0;  /* MoF - Mortar Factor */
1141    if (choice == 2) MoF = 0.9;
1142    JF = WF * DF * MoF;
1143    printf("Enter condition factor of bridge (0 -1.0):\n");
1144    scanf("%5f",&CF);
1145    if (CF < 0.4) printf(alarm_mesg);
1146    MAL = SRF * PF * MF * JF * CF * PAL;
1147    printf("Modified Axle Load = %10.2f",MAL);
1148    axle_lift_off(span,MAL);
1149    fclose(fd);

/*  SHOW RELATIONS IN A DATA BASE   */
1150  show_rels_in_db()
1151  {
1152     sprintf(param,"help\n");
1153     write(to_ingres,param,strlen(param));
1154     run_ingres();
1155  }

/*  LIST DOMAIN NAMES AND FORMATS    */
1156  list_d()
1157  {
1158     char relation[SIZE];
1159     printf("Enter name of relation:\t");
1160     scanf("%s",relation);
1161     sprintf(param,"help %s\n",relation);
1162     write(to_ingres,param,strlen(param));
```

```
1163     run_ingres();
1164  }

/*****************************************************
 PRINT OUT CONTENTS OF A RELATION
*****************************************************/
1165  print_rel()
1166  {
1167     char c;
1168     printf("Enter name of relation:\t");
1169     scanf("%s",relation);
1170     sprintf(param,"print %s",relation);
1171     write(to_ingres,param,strlen(param));
1172     run_ingres();
1173  }

/*********************************
 VIEW QUERY BUFFER
*********************************/
1174  view_buf()
1175  {
1176     write(to_ingres,"\\p\n",3);
1177  }
1178  char *store_struct (n) int n;
1179  {
1180        static char *struct_store[] =
1181        {
1182         "hash","isam","heap"
1183        };
```

```
1184        return(struct_store[n]);
1185 }
```

```
/*****************************************************************

 CREATING AND MAINTAINING A DATA BASE USING INGRES

*****************************************************************/

 1186  create_maint()
 1187  {
 1188     char com;
 1189     do
 1190     {
 1191        switch(com = menu(keytab2,STRING2,1))
 1192        {
 1193           case 'C':
 1194           case 'c':
 1195                    create_r();
 1196                    wait_for_signal();
 1197                    break;
 1198           case 'F':
 1199           case 'f':
 1200                    retrieve_into();
 1201                    wait_for_signal();
 1202                    break;
 1203           case 'D':
 1204           case 'd':
 1205                    destroy_r();
 1206                    wait_for_signal();
 1207                    break;
 1208           case 'E':
 1209           case 'e':
 1210                    erase_rel();
```

```
1211                    wait_for_signal();
1212                    break;
1213            case 'H':
1214            case 'h':
1215                    how_to_copy_rel();
1216                    wait_for_signal();
1217                    break;
1218            case 'M':
1219            case 'm':
1220                    sys_mod();
1221                    break;
1222            case 'T':
1223            case 't':
1224                    to_destroy_db();
1225                    break;
1226            case 'S':
1227            case 's':
1228                    choose_storage_structures();
1229                    wait_for_signal();
1230                    break;
1231            case 'Q':
1232            case 'q':
1233                    break;
1234            default:
1235                    printf("No such facility, Please try again.\n");
1236                    break;
1237        }
1238    }
```

```
1239     until_quit;
1240  }

/*Creates a relation with no tuples in it */
1241  create_r()
1242  {
1243     char format_t[5];
1244     int i;
1245     printf("Enter name of relation:\t");
1246     scanf("%s",relation);
1247     sprintf(param,"create %s(\n",relation);
1248     write(to_ingres,param,strlen(param));
1249     printf("Enter number of domains:\t");
1250     scanf("%d",&domain_c);
1251     for (i = 1; i <= domain_c; i++)
1252     {
1253        printf("Enter domain name:\t");
1254        scanf("%s",domain_n);
1255        printf("Enter format type:\t");
1256        scanf("%s",format_t);
1257        sprintf(param,"%s = %s",domain_n,format_t);
1258        write(to_ingres,param,strlen(param));
1259        if ( i != domain_c ) write(to_ingres,",\n",2);
1260        else write(to_ingres,"\n",1);
1261     }
1262     write(to_ingres,")\n",2);
1263     run_ingres();
1264  }
```

```
/* Creates a relation from existing relaions */
  1265  retrieve_into()
  1266  {
  1267      char n_relation[SIZE], c_domain_n[SIZE];
  1268      int i,j;
  1269      printf("Enter name of new relation:\t");
  1270      scanf("%s",n_relation);
  1271      printf("Enter number of source relations:\t");
  1272      scanf("%d",&relation_c);
  1273      for (i = 0; i < relation_c; i++)
  1274      {
  1275          printf("Enter name of source relation :\t");
  1276          scanf("%s",s_relation[i]);
  1277          range(range_n[i],s_relation[i]);
  1278      }
  1279      sprintf(param,"retrieve into %s(\n",n_relation);
  1280      write(to_ingres,param,strlen(param));
  1281      printf("Enter number of domains to be retrieved:\t");
  1282      scanf("%d",&domain_c);
  1283      for (i = 1; i<= domain_c; i++)
  1284      {
  1285          printf("Enter domain name:\t");
  1286          scanf("%s", domain_n);
  1287          printf("Enter domain name's relation:\t");
  1288          scanf("%s",temp);
  1289          for(j = 0; j<= relation_c; j++)
  1290          {
  1291              if(strcmp(temp,s_relation[j]) == 0)
```

```
1292
1293        {
1294        sprintf(param,"%c.%s",range_n[j],domain_n);
1295         write(to_ingres,param,strlen(param));
1296        if(i != domain_c) write(to_ingres,",\n",2);
1297            else write(to_ingres,"\n",1);
1298            }
1299       }
1300    }
1301          write(to_ingres,")\n",2);
1302    printf("Enter name of domain common to all relations:\t" );
1303    scanf("%s",c_domain_n);
1304    for ( i = 0 ; i < relation_c; i += 2 )
1305    sprintf(param,"where %c.%s = %c.%s",range_n[i],c_domain_n,
1306    range_n[i+1],c_domain_n);
1307    write(to_ingres,param,strlen(param));
1308    run_ingres();
1309 }

/* Destroys a relation or its contents */
1310 destroy_r()
1311 {
1312    printf("Enter name of relation:\t");
1313    scanf("%s",relation);
1314    sprintf(param,"destroy %s",relation);
1315    write(to_ingres,param,strlen(param));
1316    run_ingres();
1317 }
```

```
/* Deletes tuples in a relation */
 1318  erase_rel()
 1319  {
 1320      printf("Enter name of relation:\t");
 1321      scanf("%s",relation);
 1322      sprintf(param,"modify %s to truncated",relation);
 1323      write(to_ingres,param,strlen(param));
 1324      run_ingres();
 1325  }

/* Copies relations to INGRES */
 1326  how_to_copy_rel()
 1327  {
 1328      int i;
 1329      char filename[SIZE];
 1330      printf("Enter name of file:\t");
 1331      scanf("%s",filename);
 1332      printf("Enter name of relation:\t");
 1333      scanf("%s",relation);
 1334      sprintf(param,"copy %s(\n",relation);
 1335      write(to_ingres,param,strlen(param));
 1336      printf("Enter number of domains:\t");
 1337      scanf("%d",&domain_c);
 1338      for ( i = 1; i <= domain_c; i++)
 1339      {
 1340          printf("Enter domain name:\t");
 1341          scanf("%s",domain_n);
 1342          sprintf(param,"%s = c0",domain_n);
```

```
1343        write(to_ingres,param,strlen(param));
1344        if ( i != domain_c ) write(to_ingres,",\n",2);
1345        else write(to_ingres,"\n",1);
1346    }
1347    write(to_ingres,")\n",2);
1348    sprintf(param,"from \"%s/%s\"",PATHNAME,filename);
1349    write(to_ingres,param,strlen(param));
1350    run_ingres();
1351 }

/* Declares a variable to range over a relation */
1352 range(s1,s2) char *s1, *s2;
1353 {
1354    sprintf(param,"range of %c is %s\n",s1,s2);
1355    write(to_ingres,param,strlen(param));
1356 }

/* Modifies system relations to predetermined structures */
1357 sys_mod()
1358 {
1359    char *mesg1 = "\n\
1360    Have to be the Database Administrator!!!\n\
1361    This should be done initially after the data base\n\
1362    is created and subsequently as relations are created.0;
1363    printf(mesg1);
1364    printf("Enter name of data base:  ");
1365    scanf("%s",dbase_n);
1366    sprintf(param,"sysmod %s", dbase_n);
1367    system(param);
```

```
1368 }
1369

1370 to_destroy_db()
1371 {
1372     printf("Enter name of database you wish to destroy:\t");
1373     scanf("%s",ddbase_n);
1374     sprintf(param, "destroydb %s", ddbase_n);
1375     system(param);
1376 }

1377 choose_storage_structures()
1378 {
1379         char ch, struct_selection;
1380         int ff, ne;
1381         char *rel_help_mesg = "\n\
1382         Do you wish to know what relations are in the data base? (y/n)\n";
1383         char *store_rel_mesg ="\n\
1384         Enter name of relation whose storage structure you wish to modify\n";
1385         char *struct_selection_mesg = "\n\
1386         Which one of the following storage structures do yo wish to apply\n\
1387         to the relation\n\
1388         1. Hash   2. Isam   3. Heap\n";
1389         char *domain_help_mesg = "\n\
1390         Do you wish to know what domains are in the relation? (y/n)\n";
1391         char *dom_selection_mesg = "\n\
1392         Enter name of domain whose storage structure you wish to modify\n";
1393         printf(rel_help_mesg);
1394         if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y')
```

```
1395            {
1396                    show_rels_in_db();
1397                    wait_for_signal();
1398            }
1399        printf(store_rel_mesg);
1400        scanf("%s", relation);
1401        printf(struct_selection_mesg);
1402        struct_selection = get_number();
1403        ne = (( struct_selection -'0') - 1);
1404        printf(domain_help_mesg);
1405        if ( ( ch = yes_or_no() ) == 'Y' || ch == 'y' )
1406            {
1407                    sprintf(param,"help %s\n", relation);
1408                    write(to_ingres,param,strlen(param));
1409                    run_ingres();
1410                    wait_for_signal();
1411            }
1412        printf(dom_selection_mesg);
1413        scanf("%s",domain_n);
1414        sprintf(param,"modify %s to %s on %s",relation,store_struct(ne),
1415          domain_n);
1416        write(to_ingres,param,strlen(param));
1417        printf("Do you wish to specify a fill factor? (y/n)");
1418        if (( ch = yes_or_no() ) == 'Y' || ch == 'y' )
1419            {
1420                    printf("Enter fill factor (1 - 100):        ");
1421                    scanf("%d", &ff);
1422                    sprintf(param,"  where fill factor = %d",ff);
```

```
1423                    write(to_ingres,param,strlen(param));
1424                    run_ingres();
1425                }
1426        else run_ingres();
1427 }

1428 wait_for_more()
1429 {
1430     char s[4];
1431     fgets(s,4,stdin);
1432     return (s[0]);
1433 }

/*  YES OR NO :  */
1434 yes_or_no()
1435 {
1436    char ch;
1437    while(TRUE)
1438    {
1439       while   ( ( ( ch = getchar() ) == LF ) || ( ch == CR ) );
1440       switch (ch)
1441       {
1442          case 'Y':
1443       case 'y':
1444                return(ch);
1445       case 'N':
1446       case 'n':
1447                return (ch);
1448       default:
```

```
1449                printf("Please answer 'y' or 'n': ");
1450                break;
1451        }
1452    }
1453 }
1454 get_pathname() /* get process pathname */
1455 {
1456    FILE *fd;
1457    system("pwd > pwd_junk");
1458    fd = fopen("pwd_junk","r");
1459    fscanf(fd,"%s",PATHNAME);
1460    fclose(fd);
1461 }
    /* Determine the Axle Factor */
1462 char * axle_mesg = "  1463  axle_lift_off(s,m)
1464 {
1465    float NAF, AF, AAL;
1466    float *m, *s;
   /* No Axle Lift Off NAF is calculated from Fig 3.3, Appendix 3 */
1467    if(s <= 4.0 ) NAF = 1.0;
1468    if(s > 4 && s <= 7.5) NAF = 1.0 + 0.1428(s - 4.0 );
1469    if(s > 7.5 && s <= 14.0) NAF = 1.5 + 0.4615 * (s -7.5);
1470    if(s > 14.0) NAF = 1.75;
1471    if(s <= 2) NAF = 1.0;
1472    if(s > 2 && s <= 4) NAF = 0.68 + 0.16 * (4.0 - s);
1473    if(s > 4) NAF = 0.68;
1474    if(s <= 4) NAF = 1.0;
```

```
1475    if(s > 4 && s <= 7.2) NAF = 0.68 + 0.1 * (7.2 - s);
1476    if(s > 7.2) NAF = 0.68;
/* With Axle Lift Off, AF is obtained from Figure 3.3 see Appendix 3 */
1477    if(s <= 11.0) AF = 0.8;
1478    if(s > 11.0 && s <= 20.0) AF = 0.8 + 0.0167(s - 11.0)
1479    if(s <= 2.0) AF = 0.8;
1480    if(s > 2.0 && s <= 4.0) AF = 0.54 + 0.13 * (4.0 - s);
1481    if(s > 4.0 && s <= 10.0) AF = 0.54 + 0.0233(s - 4.0);
1482    if(s > 10.0) AF = 0.68;
1483    if(s <= 4.0) AF = 0.8;
1484    if(s > 4.0 && s <= 10.0) AF = 0.68 + 0.02 * (10.0 -s);
1485    if(s > 10.0) AF = 0.68;
1486    if(s <= 11.0) AF = 0.8;
1487    if(s > 11.0 && s <= 15.0) AF = 0.68 + 0.03 * (15.0 -s);
1488    if(s > 15.0) AF = 0.68;
1489    if(s <= 6.0) AF = 1.0;
1490    if(s > 6.0 && s <= 15.0) AF = 0.68 + 0.356 * (15.0 - s);
1491    if(s > 15.0) AF = 0.68;
1492    AAL = m * AF;
1493    printf("Do you consider that there is any axle-lift?" (y/n);
1494    printf(axle_mesg);
1495    if( (ch = yes_or_no() ) == 'Y' || ch == 'y'
1496      {
1497        AAL = m * AF;
1498      }
1499    else
1500      {
1501        AAL = m * NAF;
```

```
1502        }
1503      sscanf(DIMS,"%10f %10f %10f %10f",
1504       &single,&double,&max_gvw,&veh_type);
1505      do
1506      {
1507        sscanf(DIMS,"%10f %10f %10f %10f",
1508          &single,&double,&max_gvw,&veh_type);
1509      }
1510      while(AAL > double)
1511      MGV = max_gvw;
1512      printf("Maximum Gross Vehicle Weight = %f10.2 tonnes, MGW");
1513      printf"This is for a %s with %s axles",veh_type);
1514  }
```